Drawings are not displayable due to the volume of the data (more than 200 drawings).

* NOTICES *

1.This document has been translated by computer. So the translation may not reflect the original precisely.
2.**** shows the word which can not be translated.
3.In the drawings, any words are not translated.

CLAIMS

[Claim(s)]
[Claim 1] Two or more variable length codes interleaved by the variable-length non-encoding bit field encode. It is decryption equipment of two or more data blocks which have two or more fixed-length non-encoding fields. Said two or more variable length codes which removed two or more fixed-length non-encoding fields, and were interleaved by said the variable-length non-encoding bit field, and said variable-length non-encoding bit field, The pretreatment section which outputs two or more position signals which show the location of two or more of said fixed-length non-encoding fields in two or more data blocks, So that the decryption data of variable-length coded data inputted between said fixed-length non-encoding fields may be outputted from decryption equipment between said position signals corresponding to said fixed-length non-encoding field Decryption equipment characterized by including a means to synchronize with the data which have said position signal decrypted, and to send out to an external device.
[Claim 2] The 1st processing section which processes said two or more variable length codes which have the 1st barrel shifter set and 1st register, and were interleaved by said variable-length non-encoding bit field, Have the 2nd barrel shifter set and 2nd register, and it has further the 2nd processing section which processes two or more position signals which show the location of two or more of said fixed-length non-encoding fields in two or more data blocks. Decryption equipment according to claim 1 with which said 1st and 2nd processing section is the same with equipment, and the output and said 1st and 2nd processing section of said 1st and 2nd barrel shifter set receive the same control signal.
[Claim 3] Decryption equipment according to claim 2 with which the output of said 2nd processing section which processes the position signal which shows the location of said fixed-length non-encoding field is characterized by being used for the size decision of the non-encoding variable-length field removed from the data kept by the data register at the time of a decryption.
[Claim 4] Said pretreatment section said two or more variable length codes which removed two or more fixed-length non-encoding fields, and were interleaved by said the variable-length non-encoding bit field, and said variable-length non-encoding bit field As two or more fixed length codes which consist of two or more fixed-length bit fields And the thing for which said one fixed-length bit field was passed or removed in said pretreatment field, The front stirrup of the marker in which it has the tag in which either of having been passed in said pretreatment field is shown, and a fixed length's [tag / said] non-encoding field is shown is decryption equipment of claim 1 characterized by outputting so that it may exist back.
[Claim 5] Decryption equipment of claim 1 characterized by carrying out Huffman coding of said data block.
[Claim 6] Two or more variable length codes interleaved by the variable-length non-encoding bit field encode. It is the decryption approach of two or more data blocks which have two or more fixed-length non-encoding fields. Said two or more variable length codes which removed two or more fixed-length non-encoding fields, and were interleaved by said the variable-length non-encoding bit field, and said variable-length non-encoding bit field, The pretreatment step which outputs two or more position signals which show the location of two or more of said fixed-length non-encoding fields in two or more data blocks, So that the decryption data of variable-length coded data inputted between said fixed-length non-encoding fields may be outputted from decryption equipment between said position signals corresponding to said fixed-length non-encoding field The decryption approach characterized by including the step which is synchronized with the data which have said position signal decrypted, and is sent out to an external device.

[Claim 7] The step which processes said two or more variable length codes interleaved by said variable-length non-encoding bit field using the 1st processing section which has the 1st barrel shifter set and 1st register, The 2nd processing section which has the 2nd barrel shifter set and 2nd register is used. It has further the step which processes two or more position signals which show the location of two or more of said fixed-length non-encoding fields in two or more data blocks. The decryption approach according to claim 6 that said 1st and 2nd processing section is the same, and the output and said 1st and 2nd processing section of said 1st and 2nd barrel shifter set receive the same control signal.
[Claim 8] The decryption approach according to claim 7 characterized by having further the step which determines the size of the non-encoding variable-length field removed from the data kept by the data register at the time of a decryption according to the output of said 2nd processing section which processes the position signal which shows the location of said fixed-length non-encoding field.
[Claim 9] In said pretreatment section step, two or more fixed-length non-encoding fields are removed. Said two or more variable length codes interleaved by said the variable-length non-encoding bit field, and said variable-length non-encoding bit field as two or more fixed length codes which consist of two or more fixed-length bit fields And the thing for which said one fixed-length bit field was passed or removed in said pretreatment field, The front stirrup of the marker in which it has the tag in which either of having been passed in said pretreatment field is shown, and a fixed length's [tag / said] non-encoding field is shown is the decryption approach according to claim 6 characterized by outputting so that it may exist back.
[Claim 10] The decryption approach according to claim 6 characterized by carrying out Huffman coding of said data block.
[Claim 11] It is discrete cosine transform (DCT) equipment, interconnects with a transposed-matrix storage means and this transposed-matrix storage means, and is clocked. DCT equipment which has the arithmetic circuit which consists of a combinational circuit for performing a DCT operation, without using a storage means.
[Claim 12] DCT equipment of claim 11 with which it has the stage of a predetermined number for said combinational circuit to perform a DCT operation, and this stage is arranged sequentially.
[Claim 13] DCT equipment according to claim 11 which has a multiplexing means to multiplex the data inputted into said DCT equipment, and the output of said transposed-matrix storage means.
[Claim 14] DCT equipment according to claim 11 which has the control means which controls actuation of said DCT equipment.
[Claim 15] It is reverse discrete cosine transform (DCT) equipment, interconnects with a transposed-matrix storage means and this transposed-matrix storage means, and is clocked. Reverse DCT equipment which has the arithmetic circuit which uses the combinational circuit for performing a DCT operation, without using a storage means as the main component.
[Claim 16] Reverse DCT equipment of claim 15 with which it has the stage of a predetermined number for said combinational circuit to perform a reverse DCT operation, and this stage is arranged sequentially.
[Claim 17] Reverse DCT equipment according to claim 15 which has a multiplexing means to multiplex the data inputted into said reverse DCT equipment, and the output of said transposed-matrix storage means.
[Claim 18] Reverse DCT equipment according to claim 15 which has the control means which controls actuation of said reverse DCT equipment.
[Claim 19] It is the discrete cosine transform (DCT) approach of data, and is clocked about a DCT operation. The arithmetic circuit which considers the combinational circuit performed without a strage means as main configurations is used. The step which carries out the DCT operation of the input data according to the 1st direction, The step memorized for the transposed-matrix storage means which set the input data by which DCT was carried out in said 1st direction, and interconnected with said combinational circuit, How to have the step which carries out the DCT operation of the data memorized by said transposed-matrix storage means according to the 2nd direction using said arithmetic circuit, and obtains translation data.
[Claim 20] The DCT approach according to claim 19 which said DCT calculates on the stage of the predetermined number arranged sequentially.
[Claim 21] The DCT approach according to claim 19 of having the step which multiplexes the data inputted and the output of said transposed-matrix storage means.
[Claim 22] It is the reverse discrete cosine transform (DCT) approach, and is clocked about a reverse DCT operation. The step which carries out the DCT operation of the input loading factor according to the 1st direction using the arithmetic circuit which considers the combinational circuit performed without a strage

means as main configurations, The step memorized for the transposed-matrix storage means which doubled the input loading factor by which DCT was carried out in said 1st direction, and interconnected with said combinational circuit, How to have the step which carries out the reverse DCT operation of the multiplier memorized by said transposed-matrix storage means according to the 2nd direction using said arithmetic circuit, and obtains inverse transformation data.

[Claim 23] The reverse DCT approach according to claim 22 which said reverse DCT calculates on the stage of the predetermined number arranged sequentially.

[Claim 24] The reverse DCT approach according to claim 22 of having the step which multiplexes the data inputted and the output of said transposed-matrix storage means.

[Claim 25] The storage which memorizes claims 6-10 and an approach according to claim 19 to 24 as a program code which can perform a computer.

[Translation done.]

Drawings are not displayable due to the volume of the data (more than 200 drawings).

* NOTICES *

1.This document has been translated by computer. So the translation may not reflect the original precisely.
2.**** shows the word which can not be translated.
3.In the drawings, any words are not translated.

DETAILED DESCRIPTION

[Detailed Description of the Invention]
[0001]
[Field of the Invention] Some of codes are related with the decryption machine which is not changed about the decryption machine which decrypts the variable length code in which 0 or the variable-length bit field beyond it by which this invention is not encoded was inserted. This invention relates to the discrete cosine transform (DCT) equipment which can operate at high speed again, using a data path without a pipeline or a storage means.
[0002]
[Description of the Prior Art] Generally, a big quantity of data are compressed by various reasons including use with some stage means of variable length coding like electrical transmission, storage, read-out, and Huffman coding, and are elongated again. Huffman coding was first indicated by the paper by D.A. Huffman, and "the construction approach (1098 "A Method for the Construction of Minimun Redundancy Codes"Proc. IRE, 40: 1952) of the minimum redundancy code." In many cases, the variable-length sign in a coding bit string is discontinuous, and other non-encoding bit fields are inserted. This bit field supplies the additional matter over coded data which expresses control and/or format information and contains/or a marker header, a marker code, a staff cutting tool, a padding bit, and the overhead bit contained, for example, JPEG coded data etc.
[0003] In variable length coding, the sign of die length which is different in different input data based on the probability of generating of input data is assigned so that input code with high frequency can assign a sign shorter than data with low frequency statistically. The input code with low frequency can assign a long code. Assignment of a code is made by either statistical or the adaptation target. In statistical assignment, it is not concerned with which data block is processed, but the same output code is given to fixed data. In accommodative assignment, output code is assigned based on change by interblock [of specific input block or a data block / of a set / the statistical analysis and interblock / which are expected / (or between block sets)].
[0004] When a high-speed variable-length decryption is needed, a serious problem occurs. a problem is cut when a coded data train like especially a JPEG criterion contains the bit field of the variable length which is not encoded in which the coded data was inserted (interleave). The big difficulty in a high-speed decryption of such variable-length coded data is generated when it cannot distinguish, unless it is after a decryption of the data with which the die length of a specific non-encoding bit field continues like a JPEG criterion (it encoded) is completed completely. Since the starting position of the following coded data is not known unless it is after a decryption of a back code finishes completely, generally direct pipeline processing cannot be used with a decryption machine.
[0005] Existing solution is whether a number step (clock cycle) is needed for a decryption of one input data, or to carry out the coincidence decryption of many symbols from one in false on one stage (clock cycle) using a repeat unit (iterative units) to many applications, although it is too late. However, often the addition of the further decryption block does not balance economically, carries out such a decryption machine, and obtains and carries out still more nearly required sufficient rate. This is because two or more decryption machines do not carry out perfect juxtaposition actuation in order to be dependent on the processing result of the decryption machine located before processing initiation of the following decryption machine still determines the head of the following input data. Consequently, even if it decrypts two or more symbols on one stage (clock cycle), the stage (clock period) will be relatively long, and it will be too late in many applications as the whole decryption

machine.
[0006] Therefore, the demand to the decryption machine of the variable length code interleaved by the variable-length non-encoding bit field which solved the trouble of the conventional decryption machine one or more than it exists clearly. Specifically, the discrete cosine transform (DCT) equipment shown in drawing 77 is realized by performing 1-DDCT in the line of a 8x8-pixel block of 2 perfect-dimensional (2-D) conversion of a 8x8-pixel block first. And another 1-DDCT is performed to the train of a 8x8-pixel block. Such equipment specifically consists of an input circuit 1096, an arithmetic circuit 1104, a control circuit 1098, a transposed-matrix memory circuit 1090, and an output circuit 1092.
[0007] An input circuit 1096 receives a 8-bit pixel from 8x8 blocks. An input circuit 1096 is connected to an arithmetic circuit 1104 by the middle multiplexing machines 1100 and 1102. An arithmetic circuit 1104 performs arithmetic actuation to either a 8x8-block line or the whole train. A control circuit 1098 controls all other circuits, and realizes a DCT algorithm. the output of an arithmetic circuit is boiled and connected to the transposed-matrix memory 1090, a register 1095, and an output circuit 1092. Transposed-matrix memory is connected to the multiplexing machine 1100 which supplies an output to the following multiplexing machine 1102. The multiplexing machine 1102 receives an input from a register 1094 again. The transposed-matrix memory 1090 receives 8x8-block data in a line, and generates data in a train. An output circuit 1092 supplies the multiplier of DCT made by the pixel data block of 8x8.
[0008] In typical DCT equipment, since the arithmetic circuit is the most complicated, the rate of an arithmetic circuit 1104 determines the whole rate fundamentally. The arithmetic circuit 1104 in drawing 77 is divided and constituted by some stages to which data processing is explained below with reference to drawing 78. These stages 1114, 1148, 1152, and 1156 are realizable with an assembly in one circuit, although common use of an adder, the multiplier, etc. is carried out. However, since such a circuit 1104 constitutes two or more stages in one circuit used in common, it has the fault that it is late compared with what was optimized. This includes the storage means used for storage of the intermediate result. The time amount assigned as a clock cycle of such a circuit must be the same as the time amount of the latest stage in a circuit, or must be more than it, and it is because the time amount as the whole may become longer than the sum total of all stages.
[0009] Drawing 78 shows the typical operation data path as a part of DCT of four stages in the equipment shown in drawing 77. The function is reflected although drawing is not necessarily reflecting the actual configuration. Each of four stages 1144, 1148, 1152, and 1156 consists of circuits in which one reconstruction is possible. Four operation stages 1144, 1148, 1152, and 1156 each which are 1-DDCT(s) are reconfigurated for every cycle. In this circuit, each of four stages 1144, 1148, 1152, and 1156 is using the assembly of what is used in common (an adder and multiplier), and is minimizing hardware.
[0010] However, I hear that the fault of this circuit is late compared with what was optimized, and there is. Each of four stages 1144, 1148, 1152, and 1156 consists of assemblies of the same adder and a multiplier. A clock period is decided on the latest stage and is 20ns in block 1144 in this example. When delay (2ns each) of an input and the output multiplexing machines 1146 and 1154 and delay (3ns) of a flip-flop 1150 are added, sum total time amount is 27ns. Therefore, this DCT element can be operated in 27ns.
[0011] The pipelined DCT element is also known well. The trouble of this configuration is needing a lot of hardware for a configuration. Although offer does not carry out in this invention, the same engine performance, i.e., processing speed, the compromise of very good engine-performance pair magnitude is offered. Furthermore, a rate-advantage better than most present DCT elements is offered. Therefore, the demand to DCT / the reverse DCT approach, and equipment which can solve one or the technical problem beyond it which a Prior art has and which were improved is clear. Especially the need for the approach of shortening the time amount which the central arithmetic circuit which calculates a required result in DCT / reverse DCT equipment takes, and improving the engine performance of DCT or whole reverse DCT, and equipment is clear.
[0012]
[Means for Solving the Problem] The data encoded by two or more variable length codes by which the 1st of this invention was interleaved by the variable-length non-encoding bit field, It is decryption equipment of two or more data blocks which have the fixed-length non-encoding field which is not encoded. Two or more variable length codes which removed two or more fixed-length non-encoding fields, and were interleaved by the variable-length non-encoding bit field, and the variable-length non-encoding bit field, The pretreatment section which outputs two or more position signals which show the location of two or more fixed-length non-encoding fields in two or more data blocks, So that the decryption data of variable-length coded data inputted between fixed-length the non-encoding fields may be outputted from decryption equipment between the position signals

corresponding to fixed-length the non-encoding field It is decryption equipment including a delivery means to synchronize with the data which have a position signal decrypted, and to deliver to the output of decryption equipment.

[0013] Moreover, the 1st processing section which processes two or more variable length codes which have the 1st barrel shifter set and 1st register, and were preferably interleaved by the variable-length non-encoding bit field, Have the 2nd barrel shifter set and 2nd register, and it has further the 2nd processing section which processes two or more position signals which show the location of two or more fixed-length non-encoding fields in two or more data blocks. The 1st and 2nd processing section is the same, and the output and the 1st and 2nd processing section of the 1st and 2nd barrel shifter set are decryption equipment which receives the same control signal.

[0014] The output of the 2nd processing section which processes the position signal which shows the location of the fixed-length the non-encoding field as another desirable configuration is decryption equipment used for the size decision of the non-encoding variable-length field removed from the data kept by the data register at the time of a decryption. Furthermore, as another desirable configuration, the pretreatment section removes two or more fixed-length non-encoding fields. Two or more variable length codes interleaved by the variable-length non-encoding bit field, and the variable-length non-encoding bit field as two or more fixed length codes which consist of two or more fixed-length bit fields And the thing for which one fixed-length bit field was passed or removed in the pretreatment field, The front stirrup of the marker in which it has the tag in which either of having been passed in the pretreatment field is shown, and a fixed length's [tag] non-encoding field is shown is decryption equipment outputted so that it may exist back.

[0015] It is still more desirable that Huffman coding of the data block is carried out. Moreover, the data encoded by two or more variable length codes by which the 2nd of this invention was interleaved by the variable-length non-encoding bit field, It is the decryption approach of two or more data blocks which have the fixed-length non-encoding field which is not encoded. Two or more variable length codes which removed two or more fixed-length non-encoding fields, and were interleaved by the variable-length non-encoding bit field, and the variable-length non-encoding bit field, The pretreatment step which outputs two or more position signals which show the location of two or more fixed-length non-encoding fields in two or more data blocks, So that the decryption data of variable-length coded data inputted between fixed-length the non-encoding fields may be outputted from decryption equipment between the position signals corresponding to fixed-length the non-encoding field It is the decryption approach containing the delivery step which is synchronized with the data which have a position signal decrypted, and is delivered to the output of decryption equipment.

[0016] The step which processes two or more variable length codes preferably interleaved by the variable-length non-encoding bit field using the 1st processing section which has the 1st barrel shifter set and 1st register, The 2nd processing section which has the 2nd barrel shifter set and 2nd register is used. It has further the step which processes two or more position signals which show the location of two or more fixed-length non-encoding fields in two or more data blocks, the 1st and 2nd processing section is the same, and it is the decryption approach that the output and the 1st and 2nd processing section of the 1st and 2nd barrel shifter set receive the same control signal.

[0017] Furthermore, it is the decryption approach of having further the step which determines the size of the non-encoding variable-length field removed from the data kept by the data register according to the output of the 2nd processing section which processes the position signal which shows the location of the fixed-length the non-encoding field at the time of a decryption. In a pretreatment section step, two or more fixed-length non-encoding fields are removed preferably. Two or more variable length codes interleaved by the variable-length non-encoding bit field, and the variable-length non-encoding bit field as two or more fixed length codes which consist of two or more fixed-length bit fields And the thing for which one fixed-length bit field was passed or removed in the pretreatment field, The front stirrup of the marker in which it has the tag in which either of having been passed in the pretreatment field is shown, and a fixed length's [tag] non-encoding field is shown is the decryption approach outputted so that it may exist back.

[0018] Moreover, it is desirable that Huffman coding of the data block is carried out. Notice especially other explanation about the explanation related to drawing 82 -91 and it from the first in the following detailed explanation. The 3rd of this invention is discrete cosine transform (DCT) equipment, interconnects with a transposed-matrix storage means and a transposed-matrix storage means, and is clocked. It is DCT equipment which has the arithmetic circuit which consists of a combinational circuit for performing a DCT operation, without using a storage means.

[0019] It has the stage of a predetermined number for a combinational circuit to perform a DCT operation preferably, and a stage is arrangement DCT equipment sequentially. Moreover, it is DCT equipment which has a multiplexing means to multiplex preferably the data inputted into DCT equipment, and the output of a transposed-matrix storage means. Moreover, it is DCT equipment which has the control means which controls actuation of DCT equipment.
[0020] The 4th of this invention is reverse discrete cosine transform (DCT) equipment, interconnects with a transposed-matrix storage means and a transposed-matrix storage means, and is clocked. It is reverse DCT equipment which has the arithmetic circuit which uses the combinational circuit for performing a DCT operation, without using a storage means as the main component.
[0021] It is the discrete cosine transform (DCT) approach of the 5th data of this invention. It is clocked about a DCT operation. The step which carries out the DCT operation of the input data according to the 1st direction using the arithmetic circuit which considers the combinational circuit performed without a strage means as main configurations, The step memorized for the transposed-matrix storage means which set the input data by which DCT was carried out in the 1st direction, and interconnected with the combinational circuit, It is the approach of having the step which carries out the DCT operation of the data memorized by the transposed-matrix storage means according to the 2nd direction using an arithmetic circuit, and obtains translation data.
[0022] It is the DCT approach which DCT calculates preferably on the stage of the predetermined number arranged sequentially, and you may have the step which multiplexes the data inputted and the output of a transposed-matrix storage means. The 6th of this invention is the reverse discrete cosine transform (DCT) approach, and is clocked about a reverse DCT operation. The arithmetic circuit which considers the combinational circuit performed without a strage means as main configurations is used. The step which carries out the DCT operation of the input loading factor according to the 1st direction, and the step memorized for the transposed-matrix storage means which doubled the input loading factor by which DCT was carried out in the 1st direction, and interconnected with the combinational circuit, It is the approach of having the step which carries out the reverse DCT operation of the multiplier memorized by the transposed-matrix storage means according to the 2nd direction using an arithmetic circuit, and obtains inverse transformation data.
[0023] Notice especially other explanation about the explanation related to drawing 79, 80 and 81, and it from the first in the following detailed explanation.
[0024]
[Embodiment of the Invention]
"Table of contents"

format table 9 of C bus: data-type table 10 wishing : Symbol explanation table 11: Synthetic processing table 12: Address composition table 12for SOGCS modes A: Instruction code-ized table 13 for color space conversions: Minor operation code coding table 14 for color conversion command: Huffman and quantization table table 15 which were memorized in data cache: Fetch address table 16: Table table 17 for Huffman coding: Huffman and bank address table 18 for quantization tables: instruction word-minor operation code field table 19: instruction word-minor operation code field table 20: a instruction operand-result -- WORD table 21: instruction word table 22: an instruction operand-result -- WORD table 23: Instruction word table 24: an instruction operand-result -- WORD table 25: Instruction word-minor operation code field table 26: instruction word-minor operation code field table 27: fraction table [-- explanation [of the example of suitable and others]" -- in the suitable example The big advantage is acquired by performing a hardware raster ring by use of two independent instruction streams by the hardware accelerator. Therefore, while the first instruction stream is making the printing preparations which are the present pages, the following instruction stream can make the printing preparations which are degree pages. Hardware resources can be especially used efficiently, when a hardware accelerator can operate the rate more than an output unit.

[0025] A suitable example shows the configuration which uses 2 instruction streams. However, even if the configuration using two or more instruction streams is also possible and it takes an example in a hardware trade-off, the advantage by using more streams is acquired. By using two streams, the hardware resources of a raster image co-processor can always be concerned with preparation of the continuing page, a band, a strip, etc., also while having transmitted the present page, the band, the strip, etc. to the airline printer according to an output unit.

3.1 The common block diagram 1 of two or more stream architecture is drawing having shown typically the computer hardware configuration 201 including a suitable example. The standard host computer system which consists of the host CPU 202 connected to the host storage memory 203 through the bridge 204 is contained in the configuration 201. The host computer system is equipped with general computer system functions, such as an operating system program, application, and an information display, and the host computer system is connected to standard PCI bus 206 through the PCI bus interface 207. In addition, a PCI criterion is an industry standard known well, and is equipped with PCI bus 206 in the system carrying almost all commercial computer systems, especially the commercial Microsoft Windows (trademark) operating system. By using PCI bus 206, it becomes easy to add and use for a configuration 201 one or two or more PCI cards (for example, 209) which contain further the PCI bus interface 210, other devices 211, a local memory 212, etc.

[0026] In the suitable example, in order to make into a high speed graphics operation expressed by the Page Description Language, it has the raster image accelerator card 220. Like other PCI cards 209 etc., the raster image accelerator card (it has the PCI bus interface 221) is designed so that the host CPU 202 may operate with the gestalt of the shared memory combined gently. In addition, if required, the image accelerator card 220 can also be further added to a host computer system. A raster image accelerator card is for accelerating a lot of [complicated and] processings of operation in raster image-processing actuation, and as these actuation, it (a)-(composition b)--generalization-color-space-conversion (c)-JPEG-encodes, / it is decoded.

(d) Huffman, a run length, predicting coding/decode

(e) Hierarchical image (trademark) decode

(f) Generalization affine (image transformation g) small kernel **** operation (combo roux JON)

(h) Matrix operation (i) half toning (j) package arithmetic / memory copy operation raster image accelerator card 220 is equipped with the local memory 223 further connected to the raster image co-processor 224, and the raster image co-processor 224 starts the raster image accelerator card 220 based on the instruction from the host CPU 202. Here, as for a co-processor 224, it is desirable that it is LSI (ASIC) for specified uses. Moreover, the raster image co-processor 224 has the capacity which controls at least one printer device 226 which is the need through a peripheral interface adapter 225. Furthermore, the image accelerator card 220 can also control an input/output devices, such as a scanner. It unites, the accelerator card 220 is equipped with the general external interface 227 connected to the raster image co-processor 224, and monitoring and a test can also be performed. .

[0027] In execute mode, the host CPU 202 transmits a series of instructions and data through PCI bus 206, and performs generation processing of an image by the raster image co-processor 224. The transmitted data are stored not only in the local memory 223 but in the cache 230 in the raster image co-processor 224 or the register 229 in a co-processor 224.

[0028] Drawing 2 is drawing having shown the raster image co-processor 224 in the detail more. A co-

processor 224 is for accelerating the aforementioned processing, and consists of two or more parts under control of the instruction control section 235. In order that a co-processor may communicate with the external world, the local memory control section 236 for communicating with the local memory 223 of drawing 1 is provided. The peripheral-interface-adapter control section 237 is used for the communication link with a printer device, and uses a standard format of a centronics interface standard format, other video interface formats, etc. The peripheral-interface-adapter control section 237 is connected internally with the local memory control section 236. The local memory control section 236 and the external interface control 238 are connected through the input interface switch 252, and the input interface switch 252 is connected with the instruction control section 235. The input interface switch 252 is connected to the data cache control section 240 with the pixel organizer 246 again. The input interface switch 252 is for switching the data from the external interface control 237 and the local memory control section 236, and transmitting to the instruction control section 235 or the data cache control section 240, and the pixel organizer 246.

[0029] The external interface control 238 is provided in the raster image co-processor 224, in order to communicate with PCI bus 206 in drawing 1, and it is connected with the instruction control section 235. Moreover, in order to perform a test diagnostic or to input a clock signal and a global signal, it connects with the instruction control section 239, and it operates in harmony with a co-processor 224, and also the module 239 is equipped.

[0030] A data cache 230 operates under control of the data cache control section 240 connected. Although a data cache 230 is used in various applications, in order to store the value used recently [of the probability succeedingly used in a co-processor 224 / high], it is mainly used. As for above-mentioned high speed processing, processing of two or more data streams is mainly performed by JPEG coding / decoder 241, and the Maine data path section 242. Parts 241 and 242 are connected to juxtaposition at the pixel organizer 246 and two operand organizers 247 and 248. The stream processed from parts 241 and 242 is transmitted to the result organizer 249, and if required, processing and reformatting processing will be performed. In addition, since he wants to record the intermediate result in many cases, in addition to the data cache 230, it has the multi-you strike value (MUV) buffer 250 among organizers 249 the result with the pixel organizer 246. The result from the result organizer 249 will be outputted to the external interface control 238, the local memory control section 236, and the peripheral-interface-adapter control section 237, if required.

[0031] It is also possible to make JPEG coding / decoder 241, other two data paths called the Maine data path section 242, and "juxtaposition" connection of the further data path section (the 3rd) 243 as shown by the dotted line in drawing 2. Moreover, it is possible similarly to constitute 4 or the data path beyond it. In addition, although "juxtaposition" connection of the pass is made, note that it is that to which it does not operate to juxtaposition and only one pass operates at a stretch.

[0032] The design by whole ASIC of drawing 2 was made based on the following ideas. It is indispensable not to make the 1st also produce temporary image quality degradation small on a printing page first. In a video signal, although it is not sensed by human being's eyes even if such small image quality degradation exists, it is because small image quality degradation remains in a printing page eternally and it may come to be conspicuous with printed matter. Furthermore, since the part which is not printed [white] may be made on a page while the page is moving in the inside of a printer when delay will arise, by the time it results in a printer, it will become unsightly. Therefore, it becomes indispensable to provide high quality and a high speed with a result, and the approach depending on the rapidity of hardware is more desirable than the approach using software.

[0033] All various steps [required to perform printing processing] (algorithm) of operation are listed, and if the hardware which corresponds for every step has been arranged in the 2nd, the whole amount of hardware will become huge and will become very expensive to it. Moreover, a speed of hardware of operation fetches data required for processing, or is essentially restricted by the rate which transmits the data generated by processing. That is, a speed of operation receives constraint with the bandwidth of an interface.
 [0034] On the other hand, when the amount of the whole hardware is expressed typically, the various parts of required hardware are carrying out (a) duplication of the design of whole ASIC, and it is based on the surprising fact that (b) coincidence does not perform. Before especially this point processes data, it is notably seen in the overhead at the time of transmitting data.

[0035] It decided to reduce the amount of hardware, making all the parts of hardware as active as possible through the step of whether to be and to attach from such a viewpoint. In the 1st step, it has recognized that the repeat operation of the same fundamental class is required in many cases by image actuation. Therefore, if data

are inputted in the shape of a stream, the processing section will be constituted, a long data stream will be processed so that specific processing may be performed, and the processing section will be reconfigurated so that a processing type required for a degree after that may be suited. If a data stream is quite long, since the time amount which reconstruction takes becomes so short that it can ignore as compared with the whole processing time, its throughput will improve.

[0036] Moreover, if two or more data-processing pass is prepared, the futility of the time amount which reconstruction takes can also be excluded by reconfigurating one pass, while using other pass. That is, while the Maine data path section 242 is performing more nearly general-purpose processing, when there is JPEG coding / decode like a part 241, or an additional part 243 in other data paths, processing which specialized from those, such as entropy code modulation and Huffman coding, can be performed.

[0037] Furthermore, while advancing processing, the fetch of data and transfer to a processing part can also be performed. Moreover, while being able to attain improvement in the speed further by standardizing and unifying the data of various classification, hardware resources can also be used effectively. Therefore, the fetch of data and the overhead of the whole in connection with a transfer can be reduced.

[0038] An important thing is that a co-processor 224 is performed under control of the host CPU 202 (drawing 1) here. At this point, the instruction control section 235 generalizes control of the co-processor 224 whole. The instruction control section 235 operates a co-processor 224 with the control bus 231 called CBus (C bus). It connects with each of the module 236-250 containing the set register in each module (231 of drawing 1), and CBus231 enables actuation of the whole co-processor 224. In order to make drawing 2 legible, drawing 2 does not show the connection from the control bus 231 to each module 236-250.

[0039] Drawing 3 is drawing having shown the typical layout 260 of an available module register. As for a layout 260, the register 261 and the instruction control section 235 for whole control of a co-processor 224 are contained. The same register 262 is contained in the co-processor module 236-260.

3.2 According to the host / co-processor queuing above-mentioned architecture, it turns out that it is required to fully take coordination between a host processor 202 and the image co-processor 204. However, since the solution over this is common and it is not peculiar to above-mentioned architecture, below, it explains supposing a more general count hardware environment.

[0040] in order that a present-day computer system may perform dynamic memory allocation -- what -- it is -- the memory management technique is needed. The technique for taking a synchronization between the dynamic memory allocations and memory use by the co-processor is required of the system which has one or more co-processors. It has CPU and a special co-processor and each is sharing a series of memory groups between a general computer hardware configuration. In such a system, only CPU is a part only [in the system which can allocate memory dynamically]. When CPU allocates memory so that a co-processor may use it, a co-processor can use memory freely until the memory concerned becomes unnecessary and is released by CPU. in order [namely,] to guarantee that memory is released after a co-processor finishes use of memory -- between CPU and co-processors -- what -- it is -- a synchronization is needed. Although various solutions are shown about this synchronization, it is hard to say that it is not necessarily desirable in respect of the engine performance.

[0041] Although the problem of a synchronization is avoidable if the memory allocated statically is used, it becomes impossible to fit use of a memory resource dynamically. Although similarly it is also possible to block and keep CPU waiting until a co-processor finishes activation of processing, parallelism will be lost and the whole system performance will be sacrificed. Although use of the interrupt signal which tells termination of the processing from a co-processor is also possible, when the throughput of a co-processor is very high, it will become the overhead of big processing.

[0042] Besides the requirements for high performance, it must be flexibly coped with to dynamic memory lack in such a system. It is important to carry out the engine performance to the maximum according to many computer systems, in the system possessing much memory at max using an effective resource, although various memory size configurations are possible. Similarly, actuation sufficient despite little memory in the system of the minimum memory size configuration should be made possible, and the engine performance should deteriorate flexibly in the case of memory lack at least.

[0043] In order to solve these problems, while making system performance into max, the synchronization mechanism which adaptation-izes memory use of a co-processor dynamically to system capacity or the complexity of processing to perform is required. The suitable configuration which takes the synchronization with CPU (host) and a co-processor to drawing 4 is shown. What was used in explanation of drawing 1 is used for the reference number in drawing.

[0044] In drawing 4, CPU202 has generalized all the memory management in a system. CPU202 allocates memory 203 for use by self or the co-processor 224. The co-processor 224 has the instruction set peculiar to graphics, and can execute instruction 1022 from the memory 203 currently shared with a host processor 202. Each of these instructions can write a result 1024 in a shared memory 203, and can also read an operand from memory 203. It depends on the complexity and classification of processing for the amount of the memory 203 required for memorizing 1024 here as a result of the operand 1023 of a co-processor instruction.
[0045] CPU202 also performs processing which generates the instruction 1022 executed by the co-processor 224. In order to make parallelism between CPU202 and a co-processor 224 into max, the instruction generated by CPU202 is executed in a co-processor 224, after a queuing is carried out, as shown in 1022. Refer to 1024 for each instruction in a queue 1022 as a result of the operand 1023 in the shared memory 203 assigned by the host CPU 202 for the co-processor 224.
[0046] As shown in drawing 5, in order to perform these processings, the instruction generation section 1030, the memory management section 1031, and the queue Management Department 1032 are connected. All these modules are performed as a single process on the host CPU 202. The run command in a co-processor 224 assigns the field for 1024 as a result of the operand 1023 of the instruction which was generated in the instruction generation section 1030 and generated using service of the memory management section 1031. Moreover, the instruction generation section 1030 carries out the queuing of the instruction executed by the co-processor 224 using service of the queue Management Department 1032.
[0047] If each instruction is executed in a co-processor 224, CPU202 can release the memory currently allocated to the operands of an instruction by the memory management section 1031. The result of a certain instruction is able to serve as an operand of the next instruction, and memory is released by CPU after that. An interrupt signal is sent out, memory is not released at the same time a co-processor 224 finishes an instruction, but a clean-up device is started at a certain time after a co-processor 224 finishes an instruction, and a system releases the resource which processing of an instruction took. When a clean-up device is started, it is dependent on the relation between the memory management section 1031 and the queue Management Department 1032, and can be made dynamically adapted according to the available amount of system memories, or the amount of memory required for each co-processor instruction.
[0048] Drawing 6 is drawing having shown the configuration of the co-processor instruction queue 1022 typically. An instruction group is inserted in the pending instruction queue 1040 by the host CPU 202, is read by the co-processor 224, and is put into activation. After the executive operation in a co-processor 224 is completed, an instruction is transmitted to the clean-up queue 1041, and after a co-processor 224 finishes processing, it releases the resource which the instruction needed.
[0049] Instruction queue 1022 self is constituted as a round buffer of immobilization or dynamic adjustable size. The instruction queue 1022 has separated generation of the instruction by CPU202, and activation of the instruction in a co-processor 224. The operand and result memory of each instruction are assigned to an instruction generate time by the memory management section 1031 (drawing 5) according to the demand from the instruction generation section 1030. The memory allocation for the instruction generated newly starts coordination actuation with the memory management section 1031 and the queue Management Department 1032 which explain below, and the system enables it to be automatically adapted for the available amount of memory, or the complexity of an instruction.
[0050] The instruction queue Management Department 102 can stand by until a co-processor 224 finishes executing the instruction generated by the instruction generation section 1030. However, if the instruction queue 1022 and memory 203 which are allocated by the memory management section 1031 are large enough, it is not necessary to stand by until it is not necessary to wait for a co-processor 224 at all or and all instruction sequence is completed at least. Effectiveness is large in order that these standby times may reach also in several minutes in a big job. However, the memory usage of a peak period may exceed the available amount of memory easily. At this time, cooperative actuation is started between the queue Management Department 1032 and the memory management section 1031.
[0051] For the instruction queue Management Department 1032, "clean-up" of the ended instruction is carried out, and when the directions which release the memory allocated dynamically are made, it is proper and does not matter. When the memory with the available memory management section 1031 having decreased, or having been lost is detected, clean-up processing is directed to the queue Management Department 1032, and means to make the memory which is not used any longer by the co-processor 224 release are taken. Thereby, the memory management section 1031 can satisfy the memory requirement which the instruction newly

generated from the instruction generation section 1030 takes, without CPU's202 waiting for a co-processor 224, or synchronizing with a co-processor 224.

[0052] Even if it gives the demand which carries out clean-up of the termination instruction from the memory management section 1031 to the queue Management Department 1032, when [which is sufficient for filling the new demand of the instruction generation section] memory is not released enough, it is required that the memory management section 1031 should stand by until some instructions, for example, one half, are completed during the processing in the pending instruction queue 1040 to the queue Management Department 1032. By this, CPU202 processing will be blocked until some of co-processor 224 instructions are completed. Termination of some of co-processor 224 instructions obtains sufficient memory to release the operand of these instructions and fill a demand. By waiting for some instructions under processing, some instructions at least will exist in the pending instruction queue 1040, and the co-processor 224 will always operate. By carrying out clean-up of the part in the pending instruction queue 1040 with which CPU202 stands by, for the memory management section 1031, sufficient memory is released and, in many cases, the demand of the instruction generation section 1030 can be filled.

[0053] When a co-processor 224 is the special case where only the memory with which it fills a demand even if the one half of a pending instruction stands by until it carries out activation termination was not released, the memory management section 1031 takes the means of the last of standing by until all pending co-processor instructions are completed. as [exceed / the current memory space of a system] -- extraordinary -- size -- coming -- inside ** -- except for a complicated job etc., enough resources for this to fill the demand of the instruction generation section 1030 are released.

[0054] It becomes possible to make a throughput into max efficiently in the amount 203 of memory given to the system by coordination actuation with such the memory management section 1031 and the queue Management Department 1032. If there is more memory, the need for a synchronization decreases and can obtain a bigger throughput. On the contrary, in the case of fewer memory, it stands by more often until the processing using the memory 203 with a scarce co-processor 224 finishes, and although available memory carries out actuation at least, the engine performance deteriorates.

[0055] In case the demand from the instruction generation section 1030 is filled, the processing step which the memory management section 1031 performs is summarized to below. Each step is performed one by one and it investigates whether sufficient memory 203 for the memory management section 1031 to fill a demand is obtained after a step. Since a demand is filled when sufficient memory is obtained, a step is ended. When not obtained, it progresses to the following step, and it progresses to more excessive processing in order to fill a demand.

1. 2. Which Tries to Fill Demand with Available Memory 203 -- 4. Which Waits to Complete Some 3. Pending Instructions Which Carry Out Clean-up of All the Ended Instructions -- it Waits to Complete All Pending Instructions -- in Addition Other options can also be used, such as standing by the specific instruction with which it turns out that the part (for example, one third and 2/3) from which it differs of the pending instructions is stood by, or a lot of memory is used, in order to fill a demand.

[0056] In drawing 7, in addition to the coordination actuation between the memory management section 1031 and the queue Management Department 1032, when the fixed-length instruction queue buffer 1050 overflows, the queue Management Department 1032 can also take a co-processor 224 and a synchronization. Such a situation is shown in drawing 7 and the pending instruction queue 1040 is used as the queue of an instruction of ten die length. Since it has the number with the newest largest instruction added, the newest instruction is stored in a location 9 when a field overflows. Next, the instruction inputted into a co-processor 224 is standing by in a location 0.

[0057] When a field overflows, the queue Management Department 1032 stands by until a co-processor 224 finishes processing of a pending instruction, for example, one half. Sufficient field required for the new instruction usually inserted by the queue Management Department 1032 is released by this standby. The actuation of the queue Management Department 1032 at the time of carrying out scheduling of the new instruction is as follows.

1. whether sufficient field for the instruction queue 1040 remains, and 2. to test -- 3. which stands by until the instruction of a predetermined number with a co-processor is completed when sufficient field does not remain -- the actuation of the queue Management Department 1032 instructed to stand by that a certain instruction which inserts a new instruction in a queue is completed is as follows.

1. When there is an ended instruction which stands by until it is directed from a co-processor 224 that the

instruction was completed and which is not carried out 2. clean-up, the actuation of the instruction generation section 1030 at the time of generating the new instruction which deletes the instruction ended next from a queue is as follows.
1. The example which showed the above process of operation of having transmitted 3. co-processor instruction which generates the instruction which requires memory required for the instruction operand 1023 of the memory management section 1031, and of which 2. transfer is done to the queue Management Department 1032, and executing it, in the form of a pseudo code is shown below.
[0058]
Memory management ALLOCATE_MEMORY BEGIN IF Supposing sufficient memory to fill a demand is not obtained THEN Carry out clean-up (eradication) of all the ended instructions. ENDIF IF Supposing sufficient memory to fill a demand is not yet obtained THEN WAIT_FOR_INSTRUCTION is called. It waits for termination of the one half of a pending instruction. ENDIFIF Supposing sufficient memory to fill a demand is not yet obtained THEN An error is outputted and it returns. ENDIF The allocated memory is returned. Queue management SCHEDULE_INSTRUCTION BEGIN IF Supposing sufficient field for an instruction queue is not obtained THEN ENDIF which stands by until a co-processor ends the instruction of a certain predetermined number A new instruction is added to a queue. END WAIT_FOR_INSTRUCTION (i)
BEGIN Stand by until it is directed from a co-processor that Instruction i was completed. WHILE There is an instruction by which clean-up is not carried out although it has ended. DO IF The instruction which the degree ended is equipped with the clean-up function. THEN Call a clean-up function. ENDIF Delete the instruction ended from the queue. DONE END Instruction generation section GENERATE_INSTRUCTIONS BEGIN ALLOCATE_MEMORY is called and the instruction which allocates memory required for an instruction operand in the memory management section and to transmit is generated. Call SCHEDULE_INSTRUCTION, transmit a co-processor instruction to the queue Management Department, and execute it. END3.3 As explained in the explanatory views 1 and 3 of the register of a co-processor, a co-processor 224 is equipped with two or more registers in order to perform each instruction stream.
[0059] To the module in drawing 2, Table 1 shows the identifier of the register used in a co-processor 224, classification, and explanation, and the Appendix B explains each field of each register.
Explanation of a register [0060]
[Table 1A]
[0061]
[Table 1B]
[0062]
[Table 1C]
[0063]
[Table 1D]
[0064]
[Table 1E]
[0065]
[Table 1F]
[0066]
[Table 1G]
[0067] Those to which its attention should be paid in these registers are as follows.
(a) Instruction pointer register (ic_ipa and ic_ipb). These register pairs store the virtual address of the instruction which is carrying out current activation. The fetch of the instruction is carried out to the ascending order of the virtual address, and it performs. Jump instruction is used when moving to the virtual address where control is discontinuous. A 32-bit sequence number is given to each instruction, and a sequence number increases every [1] for every instruction to it. In co-processor 224 and host CPU 202 both sides, a sequence number is used in order to take generation of an instruction, and the synchronization of activation.
(b) Termination register (ic_fna and ic_fnb). These register pairs store the sequence number of the ended instruction.
(c) ToDo register (ic_tda and ic_tdb). These register pairs store the sequence number of the instruction by which the queuing is carried out.
(d) Interrupt register (ic_inta and ic_intb). These register pairs store the sequence number to which interrupt is applied.

(e) Interrupt status register (ic_stat.a_primed and ic_stat.b_primed). These register pairs store the prime bit which is the flag which starts interrupt, when interrupt and a termination register agree. This bit is stored like other interrupt enabling bits in an interrupt condition (ic_stat) register, or other conditions/configuration information.
(f) Register peculiarity SUSEMA forehand (ic_sema and ic_semb). The host CPU 202 must receive SEMAFOA in advance of register access which needs the rapidity to a co-processor 224, i.e., the writing to 1 times or more of a register. On the other hand, in register access which does not need rapidity, it can perform at any time. The fault which accompanies that the host CPU 202 receives SEMAFOA is that activation of a co-processor is interrupted until the instruction under current activation is completed. A register peculiarity SUSEMA forehand is constituted as 1 bit of the configuration/status register of a co-processor 224. These registers exist all over the register field of instruction control beauty. As above-mentioned, each submodule of a co-processor is equipped with the configuration/status register, respectively, and a register is set up in the usual instruction execution. All these registers are expressed on the register map, and many are implicitly corrected in an instruction execution. A host can know the contents of these registers through a register map.
3.4 In order to use a resource effective in the maximum as two or more stream format above-mentioned, and since it outputs to an external peripheral device at a high speed, a co-processor 224 performs one of the two independent instruction streams. Usually, one instruction stream supports the current output page which an output device needs at a point timely, and while the instruction stream whose 2nd instruction stream is others is stopping, it uses the module of a co-processor 224. Here, the most important point is making the most of a resource for preparation of the continuing page, a band, etc. while being outputting required output data at a point timely. Therefore, although it is completely independent, a co-processor 224 is designed so that two instruction streams (hereafter referred to as A and B) performed similarly may be performed. As for an instruction, it is desirable to be generated by the software which is operating on the host CPU 202, to be transmitted to the raster image accelerator card 220, and for a co-processor 224 to perform. At normal operation, one of the instruction streams (stream A) operates with a priority higher than other instruction streams (stream B). An instruction stream or a queue is written in one or more buffers in a host RAM 203 (drawing 1). It is a buffer at the initiation time, it is assigned, and is fixed to a host's 203 physical memory during activation of application. As for each instruction, it is desirable to be stored in a host's RAM 203 virtual memory environment, and the raster image co-processor 224 performs conversion to a physical address from the virtual address, and determines the physical address with which it corresponds in a host RAM 203 as a location of the next instruction. These instructions are stored in the local memory of a co-processor 224 one by one.
[0068] Drawing 8 is drawing showing a format of two streams A and B stored in the host RAM 203. Streams A and B -- each format is essentially the same. The easy activation model in a co-processor 224 consists of the following.
* One of streams can also have priority. two instruction virtual stream * of A stream and B stream -- * by which only one instruction is usually executed at a certain time -- It locks " carries out. * which can also carry out priority by turns in "round robin" -- one of streams -- "-- * which cannot be concerned whenever instruction-execution [of stream priority or other streams] possible, but can also be performed certainly -- * one of whose streams may be empty -- * which may be impossible as for use of one of streams -- one of streams If the instruction of consecutiveness "does not overlap" Even if * each instruction whose the instruction of * each which may include an instruction which is carrying out "overlap" to activation of the next instruction has a 32-bit "meaning" sequence number which increases one at a time has the code which stops interrupt and an instruction execution In order to make effect of delay of good * external interface into the minimum, the instruction control section 235 which may fetch an instruction beforehand In order to perform execution control of the whole co-processor 224, since [which is the need] an instruction is fetched from a host RAM 203 by the way, the instruction-execution model of a co-processor is mounted. For every instruction, the instruction control section 235 decodes an instruction, constitutes the various registers in a module through CBus231, and performs processing which makes an applicable module execute an instruction.
[0069] Drawing 9 is drawing having shown the instruction execution cycle performed by the instruction control section 235 in the easy form. An instruction execution cycle consists of the four main stages 276-279. On the 1st stage 276, it investigates whether an instruction is pending status in an instruction stream. It fetches an instruction, in being pending status, and it is 277 and 279 which decode and perform and updates 278 and a register.

3.5 Two steps must be performed on the 1st stage of decision of the present active stream.
1. decision 2. of whether the instruction is carrying out pending -- which instruction stream is fetched to a degree -- that **** -- in order to determine which instruction to be pending, investigate the following possibility.
1. a ******** [whether there is any external error condition that the instruction control section is carrying out whether the instruction control section has stopped whether it is enabling with 2. internal error or interrupt 3. pending, and that the stream of 4.A or B locks] -- 5. -- one of stream sequence numbers -- enable ******** -- 6. -- the algorithm which determines whether the instruction is carrying out pending of the pseudo code shown [whether one of streams have the pending instruction and] below based on the above-mentioned Ruhr is shown. This algorithm can be mounted as hardware through a state-transition machine using a known technique in the instruction control section 235.
[0070]
It is in not if error mode but operation mode, is not a bypass mode, either, and is in self-test mode. if A stream is locked and it is not under pause. if A stream is in operation mode, and "while the sequence number of A stream stops, an instruction exists in A stream."
Pending of the instruction is carried out. else Pending of the instruction has not been carried out. end if else if B stream is locked and it is not under pause. if B stream is in operation mode, and "while the sequence number of B stream stops, an instruction exists in B stream."
Pending of the instruction is carried out. else Pending of the instruction has not been carried out. end if else */by which the /* stream is not locked if And A stream is not stopping in operation mode, "while the sequence number of A stream stops, an instruction exists in A stream."
Pending of the instruction is carried out. else Pending of the instruction has not been carried out. end if end if else */to which the /* interface control is not working Pending of the instruction is not carried out. end if When pending of what kind of instruction has not been carried out, it will be in "spin" or an idle state until a pending instruction is found in the instruction control section 235.
[0071] The following condition is investigated in order to determine which stream is active or which stream is performed next.
1. or one of streams are locked -- 2. -- or the instruction stream which which priority is given to the stream of A and B and was performed at the end is which -- 3. -- or one of streams are working -- 4. -- the pseudo code mounted by the instruction control section is shown, and it is shown by whether one of streams have the pending instruction, and the following how the stream which becomes active next is determined.
[0072]
if A stream is locked the following stream -- A else if B stream is locked. the following stream -- B else /* -- */by which neither of the streams is locked if A stream -- sequence number of operation mode and "A stream An instruction existing in the inside of ****** or A stream" and "B stream in operation mode If it does not carry out "an instruction is existence" to B stream while the sequence number of B stream stops" The following stream is A. else if For B stream, operation mode, "a pending instruction existing in B stream, while the sequence number of B stream stops", and "A stream are in operation mode. if it does not carry out "an instruction is existence" to A stream while the sequence number of A stream stops" -- the following stream -- B else/* -- */in which an instruction exists in neither of the streams if pri=0 B low *//*A quantity -- The following stream A else if pri=1 /*A low and B quantity */ The following stream B else if pri=2or3 /* round robin */ if The last stream A The following stream B else The following stream is A. end if end if end if end if Since conditions are always changing, they need to investigate all conditions for a short time.
3.6 the fetch instruction of a current active stream -- if the following active instruction stream is determined, the instruction control section 235 fetches an instruction using the address in a corresponding instruction pointer register (ic_ipa and ic_ipb). However, when an effective instruction already exists in the prefetch buffer in the instruction control section 235, the instruction control section 235 does not fetch an instruction.
[0073] When the following conditions are fulfilled, the instruction in a prefetch buffer becomes effective.
1. The effectiveness of the contents of the prefetch buffer whose prefetch buffer is a thing from the stream as a current active stream with the same instruction in effective 2. prefetch buffer is expressed by the prefetch bit in an ic_stat register, and the bit concerned is set when the prefetch of an instruction is successful. In addition, the external writing to any registers of the instruction control section 235 makes the contents of the prefetch buffer an invalid.
3.7 If the fetch of decode and the run command instruction is carried out and they are received, the instruction

control section 235 decodes an instruction, and in order to execute an instruction, it constitutes the register 229 of a co-processor 224.
[0074] Generation of an instruction is performed with the instruction from the host CPU 202, and the instruction format used in the raster image co-processor 224 differs from the conventional processor instruction set in that it becomes a direct overhead to a host. Moreover, since an instruction is stored in a host RAM 203 and it is transmitted to a co-processor 224 through PCI bus 206 of drawing 1, the instruction should be miniaturized as much as possible. It is desirable to carry out activation initiation with the instruction with a co-processor 224 for it to be desirable and single. Moreover, in order to enable the maximum management to future modification, it is desirable to hold the flexibility of an instruction set as much as possible. Furthermore, it is also desirable that can apply the instruction executed in a co-processor 224 also to the long stream of operand data, and optimum performance is obtained. In addition, while performing decode of "a common instruction" at a high speed briefly as instruction decode "philosophy" which a co-processor 224 uses, the design is taken in so that a host system can perform fine control to actuation of a co-processor 224 also to the processing "which is not common."
[0075] Drawing 10 shows the single instruction 280 format which consists of 8 words whose each is 32 bits. Each instruction contains the type data word 282 as a result of instruction word (OPUKODO) 281 and the operand which shows the classification of an operand. The address 283-285 of three operands A, B, and C is also included with the result address 286. Furthermore, since the information about the instruction which the host CPU 202 uses is stored, the field 287 is also included.
[0076] Drawing 11 is drawing having shown the structure 290 of instruction OPUKODO 281 of an instruction. instruction OPUKODO -- 32 bit length -- it is -- main OPUKODO 291, ** OPUKODO 292, the interrupt (I) bit 293, and a part -- the decode (Pd) bit 294, the register length (R) bit 295, the lock (L) bit 296, and die length 297 are included. Explanation of each field of instruction word 290 is shown in the following tables.
[0077] OPUKODO explanation [0078]
[Table 2A]
[0079]
[Front 2B]
[0080] By setting the I bit field 293, when an instruction is completed, an instruction can be coded so that the interrupt of the activation of an instruction may be carried out and it may stop. In addition, this interrupt is called "instruction termination interrupt." a part -- the part that various modules will be microcode-ized in advance of activation of an instruction so that it may state below if the bit of the decode bit 294 is set in part and the decode bit 294 becomes operation mode in an ic_cfg register -- a decode function is offered. The lock bit 296 is used in the case of the processing which needs one or more instructions in initiation. In this case, various registers are set in advance of an instruction, and a current instruction stream "is locked" for the next instruction. If 296 [L-bit] is set, when an instruction is completed, the fetch of the next instruction will be carried out from the same stream. It is the general definition of each instruction, and the die-length field 297 is defined as the needed number of "input data items", or the number of "output-data items", and is 16 bit length. In the processing to the stream of 64,000 items or more of input data item, 295 [R-bit] is set, and input length is obtained from the po_len register in the pixel organizer 246 of drawing 2 in it. The register concerned is set just before such an instruction.
[0081] In drawing 10, the number of the operands 283-286 required for a certain instruction is adjustable according to the instruction type to be used. The following tables show the number of operands, and the definition of die length for every instruction type.
Operand type [0082]
[Table 3]
[0083] Drawing 12 shows the data word of drawing 10 to 3 operand instructions, the data word format 300 of the operand descriptor 282, and the data word format 301 to 2 operand instructions. The detail of coding of an operand descriptor is shown in the following tables.
Operand descriptor [0084]
[Table 4]
[0085] It sets to an above-mentioned table, and in the case of fixed data-address mode, a co-processor 224 fetches or calculates one internal data item, and uses this item for it as an instruction length of the operand concerned. In the case of a tile address mode, a co-processor 224 carries out the cycle of some data, and acquires the "tile effectiveness." When L bits of an operand descriptor are zero, data are short and it means that

a data item exists in operand WORD.
[0086] In drawing 10, WORD 283-286 includes the 32-bit virtual address which shows the starting position of an operand/result where own value or data of an operand is stored as a result of each operand. The instruction control section 235 of drawing 2 decodes an instruction in two steps. First, it investigates whether main OPUKODO of an instruction is effective, and an error is generated when main OPUKODO (drawing 11) is invalid. Next, by setting up various registers through CBus231, the instruction control section 235 executes an instruction and performs actuation in which it is specified as the instruction. In addition, there is also an instruction which does not have the register to set up.
[0087] The register of each module has responded to actuation and is divided into the classification of shoes. First, there is a status-register type, from other modules, it is only "read" and there are some by which "reading/writing" is carried out with the module containing a register. Next, a configuration register "is read/written in" in most externally [the type (henceforth, config1) of an eye] from a module, and only "reading" is carried out from the module containing a register. In case these registers generally store big type configuration information, such as an address value, they are used. Although the second type (henceforth, config2) of a configuration register is read from all modules and writing is made, from the module containing a register, only reading is possible. This register type is used when addressing for every bit of a register is required.
[0088] Various things exist as a control type register. Reading/writing is possible for the first type (henceforth, control1 register) by all modules (the module containing a register is also included). Control1 register is used in case control information with a big address value etc. is stored. Similarly, the second type (henceforth, control2) of a control register is set up for every bit.
[0089] The last register type (interrupt register) contains the bit which is set to 1 with the module containing a register, writes "1" in the set bit from the outside, and can be reset to zero by things in a register. A such type register is used in order to cope with interrupt/error signal from each module.
[0090] Each module of a co-processor 224 is executing an instruction, and sets c_active Rhine on CBus231 at the time of a busy condition. For this reason, the instruction control section 235 can take "OR" of c_active Rhine from each module on CBus231, and can grasp the time of an instruction being completed. The local memory control module 236 and the peripheral-interface-adapter control module 237 can execute an overlap instruction, and are equipped with c_background Rhine started in case an overlap instruction is executed. An overlap instruction is a "local DMA" instruction which transmits data between a local memory interface and a peripheral interface adapter.
[0091] The execution cycle of an overlap local DMA instruction differs from the execution cycle of other instructions. In putting an overlap instruction into activation, the instruction control section 235 investigates whether the overlap instruction is already executed. If an overlap instruction already exists, or if the overlap instruction is the unoperated mode, after the instruction control section 235 waits to complete an instruction, it will move from it to activation of the instruction concerned. If an overlap instruction does not exist and it has become operation mode, the instruction control section 235 will decode an overlap instruction immediately, will constitute the peripheral-interface-adapter control section 237 and the local memory control section 236, and will execute an instruction. If it finishes constituting a register, the instruction control section 235 will update registers (a termination register, a status register, instruction pointer, etc.), without waiting to complete an instruction in the sense of the former. At this time, if the termination sequence number is the same as an interrupt sequence number, it will only prepare the signal concerned rather than will output an "overlap instruction termination" interrupt signal. An "overlap instruction termination" interrupt signal is outputted when an overlap instruction is completed completely.
[0092] If an instruction is decoded, an instruction control section will carry out the prefetch of the next instruction, executing a current instruction. The time amount which activation of an instruction takes rather than the fetch of an instruction and decode with almost all instructions is quite longer. The instruction control section 235 carries out the prefetch of the instruction, when the following conditions are met.
1. 3. whose instruction under 2. current activation which the instruction under current activation is not stopping [interrupt or] is not jump instruction -- if the instruction control section 235 in which the instruction which is carrying out pending to 4. which can prefetch the following instruction stream exists judges that a prefetch is possible, a demand will be given to the next instruction, it will arrange to a prefetch buffer, and a buffer will be confirmed. If processing is advanced so far, the instruction control section 235 will not have nothing to carry out until the instruction under current activation is completed, and it will only perform investigating c_active on

CBus231, and c_background Rhine for termination of the instruction concerned.
3.8 After the updating instruction of the register of an instruction control section is completed, the instruction control section 235 updates a register in order to make a new condition reflect. This processing must be performed at a high speed, in order to avoid a problem contemporary with access from the outside. This high-speed update process is performed by the following procedures.
1. Suitable register peculiarity SUSEMA forehand's acquisition. When SEMAFOA is occupied by the agent of the exterior of the instruction control section 235, after an instruction execution cycle stands by and is released until SEMAFOA is released, it moves to processing.
2. Renewal of suitable register. When an instruction is not suitable jump instruction, an instruction makes an instruction pointer (ic_ipa and ic_ipb) increase by size. At the time of jump instruction, the value of a jump place is loaded to an instruction pointer. Therefore, if a sequence number is in operation mode, a termination register (ic_fna and ic_fnb) will increase.
[0093] It is appropriately updated so that the condition that a status register (ic_stat) is also new may be made to reflect. A pause bit may be set up if required. When interrupt arose, the pause to interrupt will be in a working state or an error arises, the instruction control section 235 is stopped. A pause is started by setting the instruction-stream pause bit in a status register (a_pause and b_pause). In case an instruction execution is resumed, these bits must be reset to 0.
A c_end signal is sent out on 3.1 clock-cycle time amount and CBus231, and the purport which the instruction ended is told to other modules in a co-processor 224.
4. If it is the need, send out interrupt. Sending out of interrupt is sent out at the time of the following situations.
a. When "sequence number termination" interrupt arises. Namely, when a termination register (ic_fna and ic_fnb) sequence number is in agreement with an interrupt sequence number. At this time, interrupt is prepared, a sequence number becomes operation mode, and interrupt arises. Or when encoding so that it may be the instruction of which b. termination was done at the termination time and it may carry out interrupt. In this case, an interrupt device is started.
3.9 A register peculiarity SUSEMA forehand's semantics register access SEMAFOA is the device in which two or more instruction control registers are provided with rapid access. The following are mentioned as a register which needs rapid access.
1. Instruction Pointer Register (Ic_ipa and Ic_ipb)
2. ToDo Register (Ic_tda and Ic_tdb)
3. Termination Register (Ic_fna and Ic_fnb)
4. Interrupt Register (Ic_inta and Ic_intb)
5. Pause Bit in Configuration Register (Ic_cfg)
An external agent can read all registers safely at any time. Moreover, although an external agent can write in all registers at any time, an external agent has to receive a register peculiarity SUSEMA forehand first so that the instruction control section 235 may not update the value in these registers. An instruction control section cannot update the value in an above-mentioned register, while the register peculiarity SUSEMA forehand is declared externally. Moreover, the instruction control section 235 updates all above-mentioned registers between one clock cycles, in order to maintain a high speed.
[0094] As mentioned above, if a sequence device is in operation mode, the 32-bit "sequence number" is given to each instruction. The instruction sequence number increases one by one, and is wrapped by 0x00000000 from 0xFFFFFFFF. Shortly after the writing from the outside is made by the interrupt register (ic_inta and ic_intb), the instruction control section 235 performs the following comparisons and updating.
1. Rather than the termination sequence number (value in a termination register) of the same stream, an interrupt sequence number (value in an interrupt register) will prepare a "sequence number termination" interrupt device by an instruction control section setting the "sequence number termination" preparation bit in a status register (a_primed and the b_primed bit in ic_stat), "if large" (modulo arithmetic).
2. Prepare an "overlap instruction sequence end-of-number" interrupt device rather than a termination sequence number by an overlap instruction performing [an interrupt sequence number] in the stream concerned "small", and an instruction control section setting a_ol_primed or the b_ol_primed bit in an ic_stat register, if the interrupt sequence number is the same as that of the last overlap instruction sequence number (value in ic_loa or an ic_lob register).
3. Rather than a termination sequence number, in the stream concerned, an overlap instruction is performing [an interrupt sequence number] "small", if an interrupt sequence number is not the same as that of the last

overlap instruction sequence number, as for an interrupt sequence number, a termination instruction will be shown, and an interrupt device will not be prepared.
4. If an overlap instruction is not performing [be / it / an interrupt sequence number] in the stream concerned "small" rather than a termination sequence number, as for an interrupt sequence number, a termination instruction will be shown, and an interrupt device will not be prepared.
[0095] An external agent can set the interrupt preparation bit in a status register (a_primed, a_ol_primed, b_primed, b_ol_primed bit), and can start and cancel an interrupt device independently.
3.10 Instruction control-section drawing 13 is drawing having shown the instruction control section 235 in the detail more. The instruction control section 235 contains the execution control section 305 which processes an instruction execution cycle and manages the execution control of the whole co-processor 224. The execution control section 305 manages the execution control of the whole instruction control section 235, determines instruction sequence, performs the fetch and prefetch of an instruction, and performs decode of an instruction, and renewal of an instruction control register. An instruction control section is further equipped with the instruction decoder 306. From the prefetch buffer 307, the instruction decoder 306 receives an instruction and decodes it as above-mentioned. The instruction decoder 306 also performs processing which constitutes the register in other co-processor modules, and executes an instruction. The prefetch buffer control section 307 also manages the interface between the instruction decoder 306 and the input interface switch 252 (drawing 2) while managing reading and the writing from a prefetch buffer in the prefetch buffer control section. Moreover, the prefetch buffer control section 307 also manages renewal of two instruction pointer registers (ic_ipa and ic_ipb). Access to CBus231 (drawing 2) from the instruction control section 235, the various modules 239 (drawing 2), and the external interface control 238 (drawing 2) is performed in the "CBus" mediation section 308 which performs mediation between three modular access requests. A demand is transmitted to the register section of various modules by CBus231.
[0096] Drawing 14 is drawing having shown the execution control section 305 of drawing 13 in the detail more. As above-mentioned, the execution control section manages processing of the instruction execution cycle 275 of drawing 9, and performs especially the following processings.
1. or [taking out the next instruction from which instruction stream] -- determining -- 2. -- decode of the instruction which starts the fetch of the instruction concerned and is stored in 3. prefetch buffer -- an instruction decoder -- directing -- 4. -- the prefetch of the next instruction is determined, and it starts and opts for termination of 5. instruction, and a register will be updated if 6. instruction is completed.
[0097] The execution control section is equipped with the big core condition machine 310 (it is hereafter called the central section) which manages the whole instruction execution cycle. Drawing 15 is drawing having shown central section 310 state transition diagram which manages an above-mentioned instruction execution cycle. The execution control section is equipped with instruction prefetch Boolean part 311 in drawing 14. An instruction belongs [whether the instruction which should be executed exists, and] to which instruction stream, or this part performs decision processing of **. In the transition diagram of drawing 15, initiation 312 and prefetch 313 condition receive an instruction using this information. The register Management Department 317 of drawing 14 does the monitor of the register peculiarity SUSEMA forehand of both instruction streams, and performs processing which updates all the required registers in each module. Moreover, a termination register (ic_fna and ic_fnb) is compared with an interrupt register (ic_inta and ic_intb), and the register Management Department 317 also performs processing which determines whether "sequence number termination" interrupt should be performed. Furthermore, the register Management Department 317 also performs an interrupt preliminary treatment. The overlap instruction part 318 manages the post process of an overlap instruction through management of the suitable status bit in an ic_stat register. The execution control section is equipped with the decode interface section 319 which performs the interface between the central section 310 and the instruction decoder 306 of drawing 13 further.
[0098] Drawing 16 is drawing having shown the instruction decode section 306 in the detail more. An instruction decoder performs processing which constitutes a co-processor and executes the instruction in a prefetch buffer. The instruction decoder 306 is equipped with the instruction decode sequencer 321 which consists of big condition machines which are the combination of many small condition machines. The instruction sequencer 321 communicates with the CBus dispatcher 312 which sets the register in each module. Moreover, the instruction decode sequencer 321 tells related information, such as the effectiveness of an instruction, and an overlap situation of an instruction, to the execution control section. Here, it is confirmed whether the validity check of an instruction is OPUKODO by which instruction OPUKODO is reserved.

[0099] Drawing 17 is drawing having shown the instruction dispatcher sequencer 321 of drawing 16 in the detail more. The instruction dispatcher sequencer 321 is equipped with a configuration sequencer condition machine (for example, 325 and 326) the whole module which followed the whole sequence control condition machine 324. A configuration sequencer condition machine is given to each module which should be constituted the whole module. A condition machine defines modular co-processor microprogramming as a whole. It directs that a condition machine (for example, 325) sets various registers to a CBus dispatcher using the whole CBus, and since it is processing, a module is constituted variously. Activation of an instruction must be started in order to write in a specific register. Generally the above time amount which constitutes the register of a co-processor for processing of a sequencer 321 is required for activation of an instruction. In Appendix A, the format set up by the microprogramming processing performed by the instruction sequencer of a co-processor and the instruction sequencer 321 is shown.

[0100] In fact, the instruction decode sequencer 321 does not constitute all the modules in a co-processor for every instruction. The following tables show the configuration-of-module sequence over an instruction class with modules constituted, such as an organizer 249 (RO) and the JPEG encoder 241 (JC), as a result of the pixel organizer 246 (PO), the data cache control section 240 (DCC), the operand organizer B247 (OOB), the operand organizer C248 (OOC), and the main data path 242 (MDP). In addition, modules, such as external interface control 238 (EIC), local memory control-section 236 (LMC), and instruction control-section 235 self (IC), the input interface switch 252 (IIS), and a miscellaneous module (MM), are not constituted during instruction decode processing.

[0101] Module starting sequence [0102]

[Table 5]

[0103] In drawing 17, each configuration-of-module sequencer (for example, 325) is managed so that required register access processing may be performed and a specific module may be constituted. Moreover, the whole sequence control condition machine 324 manages actuation of the whole configuration-of-module sequencer in the above-mentioned sequence. Drawing 18 is drawing which expressed the whole sequence control which starts the configuration-of-module sequencer related according to the upper table with the state transition diagram 330. In order to set various registers during modular activation, each configuration-of-module sequencer controls a CBus dispatcher, and performs processing which changes the contents of a register.

[0104] Drawing 19 is drawing having shown more the prefetch buffer control section 307 of drawing 13 in the detail. The prefetch buffer control section is equipped with the prefetch buffer 335 for storing a single co-processor instruction (6x32-bit WORD). And a prefetch buffer is equipped with one write-in port controlled by the IBus sequencer 336, and one reading port which sends out data to an instruction decoder, the execution control section, and an instruction control-section CBus interface. The IBus sequencer 336 supervises a bus protocol in connection with the input interface switch of the prefetch buffer 335. Moreover, since an instruction is fetched, it also has the address Management Department 337 which generates the address. The address Management Department 337 has the function which chooses one of ic_ipa or the ic_ipb, and is connected to the bus to an input interface switch, the function to which one of ic_ipa or the ic_ipb is made to increase based on from which stream the fetch of the last instruction was carried out, and ic_ipa and the function to store the address of a jump place in an ic_ipb register. The PBC control section 339 controls the whole prefetch buffer control section 307.

3.11 As shown in the explanatory view 13 of a module local register file, each module containing the instruction control module itself is equipped with the internal set of the register 304 mentioned above with the CBus interface control 303 shown in drawing 20, and it performs processing which updates an internal register according to the demand concerned while it receives a CBus demand. Control of a module is performed by writing in the register 304 in a module through the CBus interface 302. Which module of the instruction control section 235, an external interface control, and a miscellaneous module controls CBus, and the CBus controller 308 (drawing 13) operates as a master of CBus, and determines whether to perform writing/read-out of a register.

[0105] Drawing 20 is drawing having shown the standard configuration of the CBus interface 303 used in each module. The Standard C Bus interface 303 is equipped with the register file 304 updated through 341 with the various submodules in a module while it receives the read-out demand and write request from CBus302. Furthermore, the control line 344 which updates the memory area of a submodule including read-out of a memory area is equipped. The Standard C Bus interface 303 is served as a destination of CBus, and receives a read-out demand and write request of the memory object of a register 304 or other submodules.

[0106] The "c_reset" signal 345 sets all the registers within the Standard C Bus interface 103 to a default. However, "c_reset" does not reset the condition machine which controls an exchange of the signal between self and a CBus master. ****** therefore, "c_reset" is sent out during CBus processing -- the processing concerned -- what -- it is -- it will end in a form. "c_int" 347, "c_exp" 348, and "c_err" 349 signal is generated from the contents of module err_int and the err_int_en register based on the following formulas.
[0107]
[Equation 1]
[0108]
[Equation 2]
[0109]
[Equation 3]
[0110] A signal "c_sdata_in" and "c_svalid_in" 345 are the data/valid signal from the module in front of the inside of a module train, and a signal "c_sdata_out" and "c_svalid_out" 350 are the data/valid signal to the next module in the inside of a module train. The following are contained as a function of the Standard C Bus interface 303.
1. Monitor/updating management 3.12 of read-out / write-in management 4. submodule of read-out / write-in management 3. static test mode of read-out / write-in management 2. memory area of register Register read-out / write-in control standard CBus interface 303 receives register read-out / write request, and the bits set demand which flow on CBus. the following two kinds as a CBus instruction which a Standard C Bus interface manages -- it is .
1. As for the type A type A, other modules carry out 1, 2, 3 or 4-byte read-out / actuation to write in to the register within the Standard C Bus interface 303. In write-in actuation, a data cycle arises in the clock cycle just behind an instruction cycle. In addition, the type fields of register writing / read-out are "1000" and "1001", respectively. It investigates whether the Standard C Bus interface 303 is either [whether the instruction has pointed out the modular address by decoding an instruction, and] read-out or write-in actuation. In read-out actuation, the Standard C Bus interface 303 chooses to which register output the "c_sdata" bus 350 is connected using the "reg" field of CBus processing. In write-in actuation, the Standard C Bus interface 303 writes data in the register chosen using the "reg" field and the "byte" field. After read-out actuation is completed, a Standard C Bus interface sends out "c_svalid" 350 at the same time it returns data. After write-in actuation is completed, the Standard C Bus interface 303 sends out "c_svalid" 350, and answers.
2. As for the type C type C, the module of everybody [one] but the cutting tool in one register carries out 1 bit or actuation written in two or more bits. An instruction and data are gathered in one WORD.
[0111] The Standard C Bus interface 303 checks an instruction, and it investigates whether the instruction has pointed out the modular address. Moreover, the "reg", "byte", and "enable" field is decoded, and a required enable signal is generated. Moreover, the data field of an instruction is taken out and the taken-out data are transmitted to all four cutting tools of WORD. By this, a required bit will be written in the enabling bit in all enabling cutting tools. In this actuation, answerback is unnecessary.
3.13 Memory area read-out / write-in control standard CBus interface 303 receives memory read-out / write request on CBus. If memory read-out / write request is received, as for the Standard C Bus interface 303, a demand will investigate whether the modular address is pointed out. And a Standard C Bus interface generates the suitable address and the address strobe 344 to the submodule which performs memory read-out / writing by decoding the address field of an instruction. In write-in actuation, a Standard C Bus interface transmits the cutting tool enable signal from an instruction to a submodule.
[0112] Actuation of the Standard C Bus interface 303 decodes the type field of the CBus instruction on CBus302, and in order for data to be incorporated by the register file 304 in the following cycle or to make it transmitted to other submodules 344, it is controlled by read-out / write-in control section 352 which generates the suitable enable signal for a register file 304 and an output selector 353. If a CBus instruction is register read-out actuation, read-out / write-in control section 352 will enable an output selector 353, and will choose the right register output of "c_sdata bus" 345. If an instruction is register write-in actuation, read-out / write-in control section 352 will enable a register file 304, and then will choose data in a cycle. If the instruction is the read/write of a memory area, read-out / write-in control section 352 will generate the suitable signal 344, and will control the memory area which a module manages. A register file 304 consists of four parts of the register-select decode section 355, an output selector 353, interrupt 356, error 357, the exception 358 generation section, the AMMASUKU error generation section 359, and the register section 360 that constitutes the register of a

certain module. The register-select decode section 355 decodes the signal "ref_en" (enable [register file]) from read-out / write-in control section 352, "write", and "reg", and generates the register enable signal for enabling a certain register. According to the signal "reg" output from read-out / write-in control section 352, an output selector 353 chooses right register data for register read-out processing, and outputs them to c_sdate_out Rhine.
[0113] The exception generation sections 356-359 will generate an output error signal (349 for example, 347-362), if an error is detected during an input. The technique of calculating each output error is as above-mentioned. The register section 360 can become various types according to a demand, as it discussed, when the configuration of a register set was explained in Table 5.
3.14 On the whole, CBus (control bus) controls each module by transmitting the information for setting the register in the Standard C Bus interface of each module as the CBus configuration above-mentioned. CBus has the following two purposes so that clearly from description of a Standard C Bus interface.
1. control bus 2. which drives each module -- the access path CBus for RAM, FIFO, and the status information in each module controls a module by setting the configuration register in a module using an instruction address-data protocol. Generally, although a register is set for every instruction, correction can be made at any time. CBus collects status information and other information and accesses RAM and FIFO data from various modules by requiring data.
[0114] CBus is driven for every processing by three following either.
1. Instruction Control Section 235 (Drawing 2) at Time of Instruction Execution
2. External interface control 238 (drawing 2) at the time of target (slave) mode bus actuation activation
3. the time of an external CBus interface being constituted -- an external device -- in any case, a drive module turns into a ** module of CBus, and all other modules are possible -- wear and it becomes a module. An instruction control section performs adjustment processing of a bus.
[0115] The following tables show one definition of a CBus signal suitable for using in a suitable example.
CBus signal definition [0116]
[Table 6]
[0117] The c_iad signal of CBus is driven by the control section in two different cycles including address data.
1. Instruction cycle which a CBus instruction and the address drive on c_iad (c_valid quantity)
2. Data cycle which data drive on c_iad (write-in actuation) or c_sdata (read-out actuation) (c_valid low)
In write-in actuation, the data about an instruction are placed on a c_iad bus immediately after an instruction cycle. In read-out actuation, the target module of read-out actuation drives a c_sdata signal until a data cycle is completed.
[0118] In drawing 21, a bus contains a 32-bit instruction address-data field. This field has the following three types (370-372).
1. It is used in order that type A actuation (370) may perform read-out/writing of the register in a co-processor, or the data area of each module. These actuation is generated by the instruction control section 231 and external CBus interface which constitute the co-processor for the external interface control 238 which is performing the target mode PCI cycle, and an instruction.
[0119] In these actuation, the clock cycle just behind an instruction cycle turns into a data cycle.
2. Type B actuation (371) is used by the diagnostic mode, and access a local memory or it generates the cycle on a common interface. These actuation is generated by the external interface control and external CBus interface which are performing the target mode PCI cycle. A data cycle is sufficient at the time of the back throat of an instruction cycle, and a data cycle is worn using a c_svalid signal and it is answered to it from a module.
3. Type C actuation (372) is used in order to set each bit in a modular register. These actuation is generated by the instruction control section 231 and external CBus interface which constitute the co-processor for an instruction. In type C actuation, there is no data cycle and data are contained in an instruction cycle.
[0120] The type field of each instruction encodes the CBus processing related according to the following tables.

CBus processing type [0121]
[Table 7]
[0122] The cutting tool field is used in order to set the bit in a register. The module field is the field which specifies the address point module of the instruction on CBus. It is the field which specifies which register in a module the register field updates. An address field specifies the addresses which are the fields which specify the memory part which operates, such as RAM and FIFO. The enabling field is the field which enables the bit as

which it was chosen in the cutting tool chosen when a bit setting instruction was used. A data field contains the bit data written in the cutting tool who should be updated.
[0123] CBus includes c_active Rhine sent out when a module is among an instruction execution for every module as above-mentioned. An instruction control section can know the time of termination of an instruction based on this signal. Moreover, CBus includes c_background Rhine which operates for every module at the time of background mode with the reset for performing reset, error detection, and interrupt, an error, and an interrupt line.
3.15 In a co-processor data type and data manipulation drawing 2, in order to make brief actuation of the co-processor section 224, especially the main computation actuation in the co-processor of the JPEG encoder 241 or the main data path, a co-processor uses the data model which differentiates an external format and an internal format. An external data format is a data format which appears in the external interface of co-processors, such as a local memory interface and a PCI bus. On the contrary, an internal data format is a format which appears between the main functional modules of a co-processor 224. Drawing 22 is drawing having shown typically various input/output formats. The input external format 381 is an input format to the pixel organizer 246, the operand organizer B247, and the operand organizer C248. These organizers reformat an input external format to the input internal format 382 into which it is inputted to the JPEG encoder 241 or the main data path section 242. Moreover, these two function parts output output data by the output internal format, and the result organizer 249 changes an output internal format into the desired output format 304.
[0124] An external data format is divided into three types in the example. The first type is the "pack stream" of the data which consist of a continuation stream in which it had the channel to four for every data, and each channel consists of 1, 2, 4, 8, or a 16-bit sample. A pack stream is used in case the data changed into a pixel and a pixel, the summarized bit are expressed. Moreover, a co-processor uses big endian bit addressing in little endian cutting tool addressing and a cutting tool. Drawing 23 shows the first example of a pack stream format. Here, each object 387 consists of three channels, the 2-bit channel 0, a channel 1, and a channel 2, for every channel. Data arrangement of this format is 388. The four-channel object 395 which each data object has 32-bit WORD, and has 8 bits for every channel is shown by the next example 390 of drawing 24. The channel object 396 which has 8 bits for every channel which begins from a bit address 397 is shown by the third example 395 of drawing 25. Of course, according to application, the actual width of face and the actual number of data channels change.
[0125] The second type of an external data format is "unpack byte stream", and is a sequence which is the 32-bit WORD only whose 1 byte in each WORD is effective. The example of this format is shown as 399 of drawing 26, and only the single cutting tool 400 in each WORD is used. the further external data format -- "-- others -- it is expressed by the object classified as a" format. Generally, these data objects are data of big front molds, such as a color space conversion table and a Huffman coding table.
[0126] A co-processor uses four internal data types. The first type is a "pack cutting tool" format, and is a format which consists of 4 active cutting tools' 32-bit WORD except for the last 32-bit WORD. Example 402 of the pack cutting tool format whose WORD is 4 bytes is shown in drawing 27. The following data type shown in drawing 28 is a "pixel" format, and is a format which consists of 32-bit WORD 403 of 4 active cutting tool channels. This pixel format is interpreted as four channel data.
[0127] The following internal data type shown in drawing 29 is an "unpack cutting tool" format, and each WORD is a format which consists of one active cutting tool channel 405 and three inactive cutting tool channels. Under the present circumstances, an active cutting tool channel occupies the minimum cutting tool. other internal data objects -- "-- others -- it is classified as a" data format. The input data of an external format is changed into a suitable internal format. Drawing 30 shows the conversion gestalt from the external format 410 performed by various organizers to an input format 411. Drawing 31 shows the conversion gestalt to the external format 413 from the internal format 412 performed by the result organizer 249.
[0128] Hereafter, the processing which performs conversion is explained more to a detail. Although it is conversion to an internal format from an input data external format first, drawing 32 shows the technique used by various organizers in transform processing. Although it is the format 416 besides the exterior in the beginning, this only passes, without passing through various organizers. Next, the format 419 which the external unpack cutting tool format 417 performs unpack normalization 418, and is called an internal unpack cutting tool is generated. unpack normalization 418 processing is inactive from external unpack byte stream -- processing which removes 3 bytes is performed. Although drawing 33 shows unpack normalization processing, only one cutting tool channel has brought an effective result in the output format 419 among the inputs which

have a 4-byte channel, and signs that the mere cutting tool is outputted are shown.
[0129] In drawing 32, pack normalization 421 processing performs processing which changes the element object in the external pack stream 422 into byte stream 423. With [the size of each element of a channel] a cutting tool [below], a sample is interpolated at a 8-bit value. For example, when changing 4 bitwises per cutting tool, 4 bit value 0xN is changed into byte value 0xNN. In the case of 1 bytes or more of object, a cut-off is performed. The input object sizes supported by the stream 422 are 1, 2, 4, and 8 or 16-bit size. In addition, these are dependent on the data object in the system by which this invention is applied, or full [of WORD].
[0130] Drawing 34 shows the situation of the pack normalization 421 when the input data 422 of the three-channel object format which it has 2 (like [drawing 23 / for every data format 386]) bits for every channel is inputted. Output data are the cutting tool channel formats 423. Under the present circumstances, if required, "interpolation processing" will be performed to each channel and a 8-bit sample will be generated.
[0131] In drawing 32, a pixel stream is sent after that to the pack processing 425, the unpack processing 426, or the element selection processing 427. Drawing 35 is what showed the example of the pack processing 425, and an inactive cutting tool channel is only removed and it shows signs that the byte stream packed to 4 active cutting tools for every WORD is generated. That is, the single effective byte stream 430 is compressed into the format 431 which has 4 active cutting tools for every WORD. The unpack processing 426 is opposite processing of pack processing mostly, and an unpack cutting tool turns into the minimum cutting tool of WORD. Drawing 36 shows signs that unpack the pack byte stream 433 and a result 434 is obtained.
[0132] When drawing 37 shows element selection 427 processing and N is made into the input channel number for every unit, it is processing which chooses N element from an input stream. Unpack processing is used when generating "a prototype pixel, 437 [for example,]," In addition, a pixel channel is buried from the minimum cutting tool. The input data of a format 436 is changed by the element selection section 427, and drawing 38 shows signs that the prototype pixel format 437 is generated.
[0133] If element selection is performed, element exchange processing 440 (drawing 32) will be performed. Drawing 38 is what showed the situation of element exchange processing, replaces a selection element by the constant value stored in the internal data register 441, and shows signs that the output element 242 is generated like an example. In drawing 32, the output of the processing stages 425, 526, and 440 is sent to the rain swap processing 444. Rain swap processing is processing which multiplexes a certain lane for every cutting tool on other lanes, and also includes the processing which reproduces a certain lane on other lanes as shown in drawing 39. In the example of drawing 38, a channel 3 and a channel 1 are replaced and signs that a channel 3 is reproduced to a channel 2 and a channel 1 are shown.
[0134] In drawing 32, after the rain swap processing 444 finishes, before a data stream is re-read and moves to the duplicate processing 446, it may be stored in the multi-you strike value RAM 250. The duplicate processing 446 is processing which only reproduces a data object. Drawing 40 seems to have applied the duplicate processing 446 to pixel data, and a duplicate factor is 1.
[0135] Drawing 41 seems to have applied duplicate processing to pack cutting tool data. Drawing 42 shows processing of an organizer 249, as a result of changing data into the output external format 384 from the output internal format 383. Although transform processing shown in drawing 32 and the same processings 424, 425, 426, and 440 are included in this processing, processing of element un-choosing [451], denormalization 452, the cutting tool addressing 453, and the write-in masking 454 is further included in processing 450. The element processing 451 in which it does not choose shown in drawing 43 is reverse processing of element selection processing of drawing 37, and unnecessary data are deleted. For example, in drawing 43, only three effective channels under input are taken out and it packs to a data item 456.
[0136] The denormalization processing shown in drawing 44 carries out almost opposite actuation of the pack normalization processing 421 shown by drawing 34. In denormalization processing, processing which changes into a non-byte value each object or data item currently treated per cutting tool is performed. The cutting tool addressing processing 453 of drawing 42 performs reconstruction processing for every cutting tool required for cutting tool addressing. In an external unpack cutting tool output stream, a minimum of 2 bits of the stream address correspond to an active stream. In the cutting tool addressing processing 453, when the external unpack cutting tool is used (drawing 45), the re-map of the output stream is carried out to other channel cutting tools from one cutting tool channel. When the external pack stream is used, the re-map of (drawing 46) and the cutting tool addressing module 453 is carried out like illustration of the starting address of an output stream.
[0137] The write-in mask processing 454 of drawing 42 is shown in drawing 47. It is the processing which carries out the mask of the channel (for example, 460) with the pack stream which is not written in. It opts for

input/output data-type conversion applied based on the contents of the following data manipulation registers.
* Pixel organizer data manipulation register (po_dmr)
* The operand organizer B and an operand organizer C data manipulation register (oor_dmr, ooc_dmr)
* Result organizer data manipulation register (ro_dmr)
A setup of each data manipulation register for an instruction is made by the following two approaches.
1. In the instruction decode processing set up for co-processor itself based on the 2. current instruction set up using the standard technique written in a co-processor register just before an instruction execution, a co-processor investigates the contents of the instruction word of data, or data word, and performs processing which determines how various data manipulation registers are set up with other processings. In addition, all the combination of an instruction and an operand is not effective. There are some which have specified the operand format in some instructions. Although the result by which the instruction containing an unsuitable operand is not defined in it "" will be generated, it may end without producing an error. If the "S" bit of a corresponding data descriptor is 0, a co-processor will set a data manipulation register and will make the present instruction reflect.
[0138] Drawing 48 is drawing having shown the format of a data manipulation register. The following tables show the various bit formats in the register shown in drawing 48.
Data manipulation register format [0139]
[Table 8A]
[0140]
[Table 8B]
[0141] In one instruction each, two or more interior / external data types may be used. As a result of an operand, although combination is altogether effective, an instruction type result with semantics [a part of such combination] is generated. A concrete combination of a data type is shown in Table 9 as a result of the operand expected to each instruction. Table 9 summarizes the data type expected in the exterior/internal format.
[0142] The data type expected [0143]
[Table 9]
[0144] In addition, the symbol used in Table 9 is as follows.
Explanation of a symbol [0145]
[Table 10]
[0146] 3.16 The data normalization circuit diagram 49 shows the computer graphics processor containing the three main functional block. The three main functional block is the central graphics engine of the pixel organizer 246, the operand organizers B and C247, the data normalization section 1062 in 248, the main data path 242, or the JPEG section 241, and the programming agent 1064 in the instruction control section 235. The instruction stream 1064 to the programming agent 1064 opts for actuation of the data normalization section 1062 and a central graphics engine. For every instruction, the programming agent 1064 performs decode processing and outputs internal control signals 1067 and 1068 to other blocks in a system. Every input data word 1069, the normalization section 1062 formats data based on the present instruction, and sends out a processing result to the central graphics engine 1063 with which further processing is performed.
[0147] The data normalization section means a pixel organizer and the operand organizers B and C briefly. These organizers send out a result to a central graphics engine in JPEG coding or the main data path, after normalizing input data appropriately including a data normalization circuit. The central graphics engine 1063 operates to the data of the standard format which is a 32-bit pixel. Therefore, the normalization section performs processing which changes input data into a 32-bit pixel format. Although the input data word 1069 to the normalization section also has 32-bit width of face, you may be a format of either a pack element or an unpack cutting tool. A pack element input stream consists of the continuous object in the inside of data word [as / whose data objects are 1, 2, 4, and 8 or 16 byte width of face]. On the other hand, an unpack cutting tool input stream consists of the WORD which is 32 bits only whose 8-bit cutting tool is effective. Furthermore, the pixel data 11 generated in the normalization section consist of 1, 2, and 3 or 4 effective channels as which a channel is defined by 8-bit width of face.
[0148] Drawing 50 is drawing having shown the concrete hardware configuration of the data normalization section 1062. The data normalization section 1062 consists of the 1073 or 32 bit (FIFO) input register (REG1) of FIFO buffers, a 32-bit output register (REG2), the normalization multiplexer 1075, and a control section 1076. The input data word 1069 is latched after being stored in FIFO1073 until all input bits are changed into 1074 (REG1) at a desired output format. The normalization multiplexer 1075 is equipped with 32 combination

switch which generates the pixel latched to REG2 by choosing the bit from the value in 1074 (REG1), and the appearance force of 1073 (FIFO). That is, the normalization multiplexer 1075 considers as an input two 32-bit input WORD 1077 and 1078 shown by x [63..32] and x [31..0].
[0149] When FIFO has at least two effective data word especially in instruction processing by using such technique, the whole equipment throughput can be raised. This depends data word on the technique fetched from memory. Although diffusion or the "lap" of request data word or the object may be carried out to the input data word adjoined in a FIFO buffer, by using an input register 1074, perfect input data can be reconfigurated using the element from the contiguity data word in a FIFO buffer, and the further store and bit strip processing which are needed in advance of the main data manipulation processing stage can be excluded. When two or more data word similar type is inputted into the normalization section, such a configuration serves as a big advantage.
[0150] A control section is REG1. 1074 and REG2 It is FIFO while generating enable signal REG1_EN1080 which updates 1076, and REG2_EN[3..0] 1081. The signal which controls 1073 and the normalization multiplexer 1075 is also generated. The programming agent 1064 of drawing 49 sends out the following configuration signals to the data normalization section 1062. They are signals, such as FIFO_WR4 signal, normalization factor n [2..0], bit offset b [2..0], channel count c [1..0], and an external format (E). Input data is written in FIFO1073 by [in which an effective data exists] sending out the FIFO_WR signal 1085 for every clock cycle. When a field is not obtained, FIFO sends out the fifo_full status flag 1086. If 32-bit input data is given, it will be investigated using an external format signal whether an input is a pack stream format (E= 1) or you are an unpack cutting tool (E= 0). In the case of E= 1, a normalization factor serves as each element size of a pack stream. Namely, in n= 0, a 4-bit width-of-face element and n= 3 show a 8-bit width-of-face element, and, as for the element of 1-bit width of face, and n= 1, n> 3 shows a 16-bit width-of-face element, as for a 2-bit width-of-face element and n= 2. Moreover, a channel count is the maximum number of the continuous input object formatted for every clock cycle, in order to generate a pixel with the number of request valid bytes. Specifically, a pixel only with the minimum cutting tool effective [c= 1], a pixel with a minimum of 2 bytes effective [c= 2], a pixel with a minimum of 3 bytes effective [c= 3], and c= 0 are all pixels with effective 4 bytes.
[0151] When a pack stream consists of the element below 8-bit width of face, the data-processing starting position in x [31..0] whose bit offset is the value stored in REG1 is determined. When bit offset is the deviation from the maximum bit which is the first input cutting tool, the generation method of output data byte y [7..0] is given by the following formulas.
In the case of n= 0, it is y[i] =x [7-b]. In the case of n= 1, it is y[i] =x at the time of 0<=i<=7 [7-b]. It is y[i] =x at the time of i= 1, and 3, 5 and 7 [6-b]. In the case of n= 2, it is y[3] =x[7-b] at the time of i= 0, and 2, 4 and 6.
y[2]=x[6-b]
y[1]=x[5-b]
y[0]=x[4-b]
y[7]=y[3]
y[6]=y[2]
y[5]=y[1]
y[4]=y[0]
In the case of n= 3, it is y[i] =x [i]. In the case of n> 3, it is y[7...0] =x [15...8] at the time of 0<=i<=7.
The same is said of the formula which generates output data byte y [15..8], y [23..16], and y [31..24].
[0152] In addition, the above technique inputs the element of an input stream, and by performing duplicate processing of a required count and generating the output object of standard width of face, it can extend it so that the output array of any die length can be generated. Moreover, a little endian or a big endian is sufficient as the order of processing of an input element. In addition, in the above-mentioned example, since processing always begins from the maximum bit which is an input cutting tool, the order of a big endian element is used. To use the order of a little endian, it is necessary to redefine bit offset as a value over an input cutting tool's minimum bit. Moreover, when input element width of face is more than standard output width of face, an output element is generated by [which omit an input element] generally deleting a suitable number of the minimum bits. By the upper formula, by choosing the maximum cutting tool of a 16-bit data object, a 16-bit input element is omitted and 8-bit width-of-face standard output is generated.
[0153] The control section of drawing 50 performs decode of n [2..0] and c [1..0], and generates the enable signal for the selection signal and REG1 for a normalization multiplexer, or REG2 using these and b [2..0].

Moreover, as for a control section, for a certain reason, it is also equipped with the counter which memorizes current bit-position in_bit [4..0] which chooses input data into REG1, and current byte-position out_byte [4..0] which begins to write in output data that FIFO comes during an instruction in the sky. When processing is completed, a control section detects input WORD by comparing the value of in_bit [4..0] with the location of the last object of REG1, and starts FIFO read-out actuation by sending out a FIFO_RD signal in one clock cycle whose FIFO is not empty. Signal fifo_empty and fifo_full are FIFO status flags, and when FIFO does not have effective data and fifo_empty=1 and FIFO are full, they are set to fifo_full=1. In the clock cycle to which FIFO_RD was sent out, REG1_EN is sent out and new data are incorporated by REG1. The enable signal of REG2 has four each to each cutting tool of an output register for every correspondence. the processing in c [1..0] by which the control section was decoded, and REG1 -- taking the minimum value in the inside of three values of the number of intact channels in the waiting number of effective elements, and REG2 -- it is -- REG2_ -- EN [3..0] is calculated. In the case of E= 0, only one effective element exists in REG1. When equal to c [3..0] by which the number of channels which occupies REG2 was decoded, perfect output WORD is obtained.
[0154] In the suitable example of this invention, the circuit field which the equipment of drawing 50 occupies can be sharply reduced by adding the functions to restrict a bit offset parameter, such as using a control section and a part of offset which are used in a normalization multiplexer. This offset limit function operates according to the following formulas depending on a normalization factor.
[0155]
b_trunc[2...0] =0 In the case of n>=3 = b [2...0] In the case of n= 0 = b [2...1] In the case of n= 1 =b [2] &"00" In the case of n= 2 ("&" shows the joint processing for every bit)
By such processing, it sets to drawing 50, and is MUX0 and MUX1... The size of each normalization multiplexer shown by MUX31 is reduced to the maximum size 20-1 when performing 32-1 to the bit offset limit when not using a limit function. Improvement in a circuit rate can also be aimed at by this size contraction.

[0156] As mentioned above, in the suitable example, it has the efficient circuit which changes data into some normalization formats.
3.17 In image-processing actuation drawing 2 and Table 2 of an accelerator card, the instruction control section 235 "executes" the instruction which can result in the actuation performed in a co-processor 224. The instruction executed includes various instructions with which a useful function is performed in the main data path section 242. One of the useful instructions of these is synthetic processing.
[0157] 3.17.1 A composite map 51 is drawing having shown the synthetic model mounted in the main data path section 242. Generally a synthetic model 462 contains the three data input sources and output data (sink) 463. One of the input sources is the pixel data 464 from the same phase hand within an output 463 and memory. Moreover, the instruction operand 465 used as data sources, such as a color and opacity, is included. Here, any of a flat, a blend, a pixel, and a tile is sufficient as a color or opacity. In addition, since it is more nearly high-speed to generate inside about a flat or a blend rather than it fetches through an input/output, it is generated in the blend generation section 467. Furthermore, input data also contains the attenuation data 466 which attenuate operand data 465.
[0158] As mentioned above, pixel data usually consist of four channels each channel of whose is 1-byte width of face. Here, 1 byte of the highest address is an opaque channel. In addition, please refer to standard reports, such as a description paper "Thomas Porter and Tom Duff"Compositing Digital Images"in Computer Graphics, volume 18, number 3, July 1984", about actuation and usefulness of synthetic processing.
[0159] A co-processor can also use pre multiplication data. Pre multiplication is processing which carries out the multiplication of each color channel and the opaque channel beforehand. Therefore, it can have the pre multiplication sections 468 and 469 of two options, and the outputs 472 and 473 which are need and by which pre multiplication was carried out by by the way carrying out the pre multiplication of the opaque channels 470 and 471 and the color data can be obtained. The synthetic section 475 compounds two inputs based on current instruction data. A synthetic operator is shown in following Table 11.
[0160] Synthetic actuation [0161]
[Table 11]
[0162] Here, (aco, ao) express the pre multiplication pixel of a color ac and Opacity ao. R is an offset value and "wc(s)" is wrapping / clamping operator who explains below. In addition, note that the synthetic section 475 is equipped also with the reverse action of each operator of an upper table. Clamp/wrapping section 476 is a clamp or the processing section for carrying out a lap about data in limiting value 0-255. Moreover, if required, data

can also be processed "ampul multiplication" of option 477, and it can also return to the pixel value of a basis. Finally, output data 463 are generated and it is returned to memory.

[0163] In case drawing 52 performs synthetic processing, it shows the instruction format sent to the main data path section. If X field in main OPUKODO is 1, an addition operator will be applied according to the aforementioned table. If this field is 0, other instructions of those other than an addition operator will be applied. Pa field is the field which shows whether the pre multiplication of the first data stream 464 (drawing 51) is carried out. Moreover, it is shown whether Pb field carries out the pre multiplication of the 2nd data stream 465, and it is shown whether Pr field carries out the "ampul multiplication" of the result using a part 477. C field shows [a lap or a clamp, overflow, or] whether an underflow is carried out in the range 0-255, and it is shown which operator the "com-code" field applies. An addition operator can also use an offset register (mdp_por). This offset is lengthened from the result of addition actuation, before wrapping / clamping processing is performed. In an addition operator, the com-code field turns into the field which shows whether it enables for every channel of an offset register.

[0164] It is made to change the standard instruction word coding 280 of drawing 10 described previously for a synthetic operand. Since the phase hand of output data is the same as the source of a basis, Operand A becomes always the same as that of result WORD. Therefore, Operand A can describe Operand B for a long time with Operand B. Like other instructions, A descriptor under instruction describes an input format, and R descriptor specifies an output format.

[0165] Drawing 53 shows the first example of an instruction word format of a blend instruction as 470. Blend processing is prescribed by the starting value 471 and exit value 472 for every channel. Similarly, drawing 54 shows the tile address 476, the initiation offset 477, and the tile instruction format specified with die length 478. All tile addresses and sizes are specified for every cutting tool. Tile processing is performed in modular one and drawing 55 is drawing explaining the fields 476-478 of drawing 54. The tile address 476 specifies the tile length of the whole to which the tile length 478 does the lap of the first cutting tool by whom the tile initiation offset 477 is used in the starting address of tile memory at the time of tile initiation.

[0166] A color element and opacity may be attenuated by the attenuation value 466 in drawing 51. An attenuation value is acquired by the following three technique.

1. By putting in an attenuation factor into the operand C WORD of an instruction, software can specify flat attenuation.

By ON, 2.1 can use the bit map attenuation with off 0 using the software which specifies the address of a bit map in the operand C WORD of an instruction.

3. Cutting tool map attenuation may be prepared in the cutting tool map address of the operand C WORD of an instruction.

4. ON and bit map attenuation made off at the time of 2 can be performed using the software which carries out a law at the time of 1.

[0167] Since an attenuation value is the integer of 0-255 without a sign, the color channel by which pre multiplication was carried out is calculating Coa=CoaxA/255, and multiplication is carried out to an attenuation factor. Here, A is an attenuation factor and the color channel to which the pre multiplication of the Co was carried out.

[0168] 3.17.2 In color space conversion instruction drawing 2 and Table 2, the main data path section 242 and a data cache 230 mainly process color conversion. A color space conversion performs transform processing to the first color space format [second] (for example, format suitable for CYM or CYMK printing) from a color space format (for example, format suitable for a RGB color display). Color space conversion processing is designed so that all color spaces may be supported, and through CBus231, the instruction control section 235 which can be used in what kind of function from one dimension to many dimensions constitutes an organizer 249 as a result of the main data path section 242, the data cache control section 240, the input interface switch 252, the pixel organizer 246, the MUV buffer 250, the operand organizer B247, and the operand organizer C248, and controls him to operate in a color translation mode. In this mode, the input image which consists of two or more lines of a pixel is sent out to the main data path section 242 for every 1-pixel Rhine as a pixel stream. The main data path section 242 (drawing 2) performs color space conversion processing for a pixel stream for every [reception and] pixel through the pixel organizer 246 from the input interface switch 252. Moreover, an interval table and a fraction table are beforehand loaded to the MUV buffer 250, and a color conversion table is loaded to a data cache 230. The main data path 242 accesses these tables through the operand organizers B and C, for example, changes a pixel into CYM or a CYMK color space from a RGB color

space, and sends the changed pixel to the result organizer 249. The main data path section 242, a data cache 230, the data control section 240, and other above-mentioned devices are the bases of control of the instruction control section 235, and operate in one of the modes of a general single output color space (SOGCS) translation mode or general two or more output color space (MOGCS) translation mode. the detail of the data cache control section 240 or a data cache 230 -- being related -- "a data cache control section and a cache" -- please refer to 240 or 230 (drawing 2) items.

[0169] Exact color space conversion processing is complicated nonlinear processing. For example, although the color space conversion processing to the single main color element (namely, cyanogen) of a CYMK color space from a RGB pixel is linearity theoretically, nonlinearity will arise in the output device which mainly outputs the color element of a pixel in fact. Also in the color space conversion processing to other main color elements (yellow, a magenta, black) of a CYMK color space from a RGB pixel, it is the same. That is, in order to compensate the nonlinearity produced in each color element, generally a nonlinear color space conversion is used. A transfer function complicated for the nonlinearity of such complicated color transform processing is incorporated, or a look-up table is used. For example, when the input color space of a 24-bit RGB pixel is given, the look-up table which maps these pixels to the 8-bit main color element (cyanogen) of a CYMK color space needs 16 megabytes or more. Similarly, the look-up table which maps a 24-bit RGB pixel to the four 8-bit main color elements of a CYMK color space becomes 64 megabytes or more, and is ** which needs a huge capacity. On the other hand, the main data path 242 (drawing 2) makes a coarse output color value correspond to the point in an input color space using the look-up table stored in the data cache 230, and a middle output is obtained by interpolating an output color value.

a. a general single output color space (SOGCS) translation mode -- consist of single and the 24-bit pixel in which a RGB color space has 8-bit red, green, and a blue element for two or more output color translation mode (SOGCS) and both sides (MOGCS). Each RGB dimension of a RGB color space is divided at the section of 15, and the die length of each section is set up so that it may become the inverse function of the nonlinearity to a CYMK color space from RGB of a printer. That is, when nonlinearity with a strong transfer function is shown, the die length of the section is shortened, and when a transfer function is close to linearity, the die length of the section is lengthened. In order to know the nonlinear part of such a transfer function, it is desirable to investigate the color space of each output printer correctly. However, it is also possible for it to be based on the description by which know-how and a printer type (for example, ink jet) were measured, and to approximate or model a transfer function. The location in the section of 15 of a color element value is decided for every color channel of an input pixel. In order to determine in which section an input color element value exists, two tables for determining the location within the section when an input color element value exists are used in the main data path section 242. Of course, a different table to the output printer which has a different transfer function may be used.

[0170] Each dimension of RGB is divided as mentioned above at the section of 15. That is, the RGB color space has three-dimension lattice structure divided in the section, and the input pixel of the both ends of the section serves as coarse arrangement in the input color space. Furthermore, only the output color value of the output color space corresponding to the both ends of the section is stored in the look-up table. Therefore, the output color value of an input color pixel determines the output color value corresponding to the both ends of the section when an input pixel exists, and is calculated by interpolating an output color value based on the location within the section. The need that mass memory must be used can be reduced by this technique.

[0171] Drawing 56 shows Example 480 which determines the location within the corresponding section or the section to the input RGB color pixel. Transform processing is performed using the section table 482 or the location table 483 within the section for every 8-bit input color channel of a 24-bit input pixel. In drawing 56, although the 8-bit input color element 481 displays 4 of a decimal number in a binary format, this 8-bit input color element 481 is used as a lookup to a section table or the location table within the section. The section table 482 outputs one of the sections from 0 to 14 when the input color element value 481 exists by 4 bits. Similarly, the table 482 within the section shows the location within the section when the input color value element 481 exists. The table within the section stores the 8-bit value of the range from 0 to 255, and this value is interpreted as a fraction of 256. Therefore, in the case of the input color value element 481 which was binary and expressed the decimal number 4, an output value 0 is generated by carrying out the lookup of the section table 482. Moreover, the output value 160 showing fractions 160/256 is generated by carrying out the lookup of the input value 4 on the location table 483 within the section. Section length is not uniform as shown in the section table 482 and the location table 483 within the section. As mentioned above, section length is decided by the

nonlinearity of a transfer function.
[0172] Three section outputs and location outputs within [of three] the section are obtained by using a section table and the location table within the section to each RGB color element as above-mentioned. The section / location table within the section to each color element are loaded to a MUV buffer (drawing 2), and when required, it is accessed by the main data path 242. The configuration of the MUV buffer 250 in color transform processing is shown in drawing 57. The MUV buffer 250 (drawing 57) is divided into three fields 488, 489, and 490 corresponding to each color element in each. Each field (for example, 488) is divided into a 4 more-bit section table and the location table within [of 8 bits] the section. The 12-bit output 492 is taken out from the MUV buffer 250 by the main data path section 242 for every input color channel. A 12-bit output is set to 000001010000 in the above-mentioned example of the single input color element of a decimal number 4.
[0173] Drawing 58 is drawing having shown the example of interpolation processing. Interpolation processings are the processings with the main interpolation to other color spaces (for example, CMY or CMYK) from one three-dimension space 500 (for example, RGB color space). From a pixel P0, P7 exists coarsely in a RGB input color space, and has the output color value valve flow coefficient (P0) to valve flow coefficient (P7) which corresponds in an output color space. From a pixel P0, the output color element value of the input pixel Pi located among P7 is the following, and is made and determined. First, the both ends P0, P1, ..., P7 of the section which encloses the input pixel Pi are determined. Next, location element frac_r within the section, frac_g, and frac_b are determined, and between valve flow coefficients (P7) is interpolated at the last using the location element within the section from the output color value valve flow coefficient corresponding to the both ends of P0 to P7 (P0).
[0174] Interpolation processing performs 1-dimensional interpolation of the direction of red (R) first, and the value of temp11, temp12, temp13, and temp14 is calculated from the following formulas.
temp11=CV(P0)+frac_r(CV(P1)-CV(P0))
temp12=CV(P2)+frac_r(CV(P3)-CV(P2))
temp13=CV(P4)+frac_r(CV(P5)-CV(P4))
temp14=CV(P6)+frac_r(CV(P7)-CV(P6))
Next, interpolation processing calculates temp21 and temp22 using the following formulas, and calculates 1-dimensional interpolation of the direction of green (G).
[0175]
temp21=temp11+frac_g(temp12-temp11)
temp22=temp13+frac_g(temp14-temp13)
Finally, the last color output value is calculated based on the following formulas, and the last dimension interpolation of the direction of blue (B) is performed.
final=temp21+frag_b(temp22-temp21)
Also when the range of an input and an output is not in agreement, it is often possible. Here, if the output range is narrower than the input range, the range must be clamped at both ends in many cases. That is, when the color around the edge of the range is changed, the strain which is not desirable arises in many cases. Drawing 59 explains the example which this problem produces, and signs that a 1-dimensional input range value is mapped in an output range value are shown. Here, the output value over an input value shall have become settled at points 510 and 511. Supposing the greatest output value is clamped at a point 512, a point 511 must be the output of this magnitude. Therefore, in interpolating two points, 510 and 511, a line 515 turns into a interpolation line and an output value 517 corresponds to an input point 516. However, when constraint of the range does not exist and an output value becomes a point 518, this technique is not necessarily the optimal color mapping. The interpolation line of 510 and 518 generates an output value 519 to an input point 516. When printing the color around the edge of the range, in order that especially the difference between such two output values 517 and 519 may avoid this problem used as the strain often attached to an eye, the main data path section is calculated in an extended output color space, is used for the following formulas, and can also be clamped [a scale or] in the suitable range.
[0176]
In drawing 58, either the SOCGS translation mode from which interpolation processing changes a RGB pixel into a single output color element (for example, cyanogen), or MOGCS mode in which a RGB pixel is changed into all output color elements at coincidence is performed. When color conversion is performed to each pixel in an image, color conversion of several 1 million pixels will be carried out independently, respectively. Therefore, in order to operate at a high speed, it is desirable to find quickly eight values (P0-P7) of the input-

value circumference.
[0177] The main data path section 242 takes out the 12-bit output which consists of a 4-bit section part and the location part within the 8-bit section for every color input channel as explained in drawing 57. The main data path section 242 combines red, green, and the 4-bit section part of a blue channel, and generates 12 single bit addresses (IR, IG, IB) like 520 in drawing 60. Drawing 60 is the data flow diagram having shown signs that the single output color element 563 was obtained from single 12 bit address 520. Twelve bit addresses 520 are first sent to the address-generation section of a data cache control section 240 like the generation section 1881 (drawing 141), and generate eight 9 bit lines / byte addresses 521 to a memory bank (B0, B1, ..., B7). A data cache (drawing 2) is divided into eight independent memory banks 522, and addressing of each is independently carried out by eight Rhine/byte addresses. Conversion to eight lines / byte address from 12 bit addresses 520 in the address-generation section is performed according to the following tables.
[0178] Address composition in SOGCS mode [0179]
[Table 12A]
[0180] Here, BIT [8:6], BIT [5:3], and BIT [2:0] show 6 to 8 of the 9-bit bank address bits, 3 to 5 bits, and 0 to 2 bits, respectively. Moreover, R [3:1], G [3:1], and B [3:1] show even the 1st to 3rd [of the 4 bit sections IR, IG, and IB of 12 bit addresses 520] bit. Mapping to 9 bits is explained to a detail from 12 bits about the memory bank 5 of Table 12. 1 of the 4-bit red section Ir in 12 bit addresses 520 - a triplet are mapped by 6-8 bits of 9 bit-address B5, and 1 of the 4-bit green section Ig - a triplet are added, it is mapped by 3-5 bits of 9 bit-address B5, and 1 of the 4-bit blue section Ib - a triplet are mapped by 0-2 bits of 9 bit-address B5.
[0181] Eight Rhine / byte addresses 521 are used as the address to the corresponding memory bank 522 which consists of 512x8 bits, and the corresponding 8-bit output color element 523 is latched from each memory bank 522. According to this addressing processing, the output color values valve flow coefficient (P0)-valve flow coefficient (P7) corresponding to endpoints P0-P7 may serve as the different address in the inside of a memory bank. For example, 12 bit addresses 0000 0000 0000 is 000 in all banks. 000 Although the same bank address 000 is obtained, they are 12 bit addresses 0000. In the case of 00000001, in banks 7, 5, 3, and 1, it is the bank address 000. It is set to 000000 and is the bank address 000 in banks 6, 4, 2, and 0. 000 The bank address which is different so that it may be set to 001 is obtained. Thus, eight single output color values valve flow coefficient (P0)-valve flow coefficient (P7) which enclose an input pixel value are acquired from each memory bank by coincidence, and it can prevent an output color value becoming a duplex in a memory bank.
[0182] Drawing 61 shows the configuration of the memory bank of a data cache 230 used in a single color translation mode. Each memory bank consists of a 128-line entry, and each Rhine entry is constituted from the 4x8-bit memory 533-536 by 32 bit length. On a memory address 521, the data stream to which it corresponds in a memory address is determined, and 7 bits is used in order to take latch 542 as a memory bank output. It is used in order to determine whether 2 bits is a byte address, becomes an input to a multiplexer 543, and takes selection 544 by considering which 4x8-bit entry as an output the bottom. The data for each eight memory banks are outputted for every clock cycle, and it is sent to the main data path section 242. That is, a data cache control section outputs the 8-bit output color value for reception and interpolation processing [in / for a 12-bit byte address / the main data path section 242] to the operand organizers 247 and 248 from the operand organizer 248 (drawing 2).
[0183] In drawing 60, the main data path section 242 (drawing 2) performs interpolation processing at three steps. In the 1st step in the main data path section, multiplication/adder unit (for example, 550) considers as an input the color value and the red section location element 551 which are outputted from a corresponding memory bank (for example, 522), and calculates four output values according to the 1st step of the aforementioned formula. The output (553 for example, 554) of the 1st step is sent to the 2nd step 556, and calculates an output 558 according to the type before the 2nd step using the frac_g input 557. Finally, the 2nd step outputs 558 and 559 and the frac_b input 562 are used, and the final output color 563 is calculated based on a front type.
[0184] The processing shown in drawing 60 is pipelined in order to obtain the throughput greatest on the whole. Furthermore, the technique of drawing 60 is used when the single output color element 563 is required. For example, the technique of drawing 60 is used, when generating the cyanogen color element of an output image first, reloading the cache table during pass after that and generating the magenta of an output image, yellow, and a black element. Especially this is suitable for 4 pass printing processing in which each output color serves as independent pass.
b. Although general two or more output color space mode co-processor 224 also performs actuation with

MOGCS mode, MOGCS mode operates almost like SOCGS mode except for some points. In MOGCS mode, the main data path section 242 of drawing 2, the data cache control section 240, and a data cache output the four main color elements cooperated and outputted to coincidence. Although for that the size of a data cache 230 is needed 4 times, in order to save a storage region, by the MOGCS mode of operation, the data cache control section 240 stores only one fourth of all the output color values of an output color space. The remaining output color values of an output color space are stored in the external memory of a low speed, and when required, they are taken out. In addition, this equipment and technique are based on the surprising fact that the rate of a mistake of the coarse color translation table in cache system is very small. This is based on the knowledge that distribution of the color value of one pixel and other pixels is small, by many color pictures. Moreover, a coarse output color value has the very high probability which becomes the same also in a neighboring pixel.

[0185] Drawing 62 shows the technique to which a co-processor performs multiple channel cache color conversion. After each input pixel is decomposed into a color element, a corresponding section table value (drawing 56) is determined as mentioned above, and the three 4-bit sections, such as Ir, Ig, and Ib570, are obtained. The combined 12 numbers of bits 570 are changed according to the above-mentioned table 12, and eight bit addresses [nine] are obtained. Remapping is carried out, the lookup of the corresponding memory bank 573 is carried out, and four color output channels 574 are obtained so that the address (for example, 572) may be explained below in drawing 63. Although a memory bank 573 can serve as a 512x32-bit entry on the whole, it stores the 128x32-bit entry of them. A memory bank 573 is used as a cache so that nothing and drawing 63 may explain a part of data cache 230.

[0186] Drawing 63 shows signs that remapping of the 9-bit bank input 578 is carried out to 579, and can remove the alias of a memory pattern by replacing the sequence of bits 580-582. Thereby, the probability of the cache element with the adjoining same pixel value for an alias to be carried out can be reduced. The reconfigurated memory address 579 is used as the address to the corresponding memory bank (for example, 585) which consists of 128 entries whose each is 32 bits. By accessing memory 585 using 7 bit-line addresses, the output carried out latch 586 for every memory bank is obtained. Each memory bank (for example, 585) has the related tag memory which consists of 128 entries whose each is 2 bits. Seven bit-line addresses are used also in order to access the tag with which it corresponds in this tag memory 587. By measuring a maximum of 2 bits of the address 579 with the tag with which it corresponds in the tag memory 587, it is determined whether the output color value is stored in the cache. A maximum of 2 bits in these 9 bit addresses are equivalent to the maximum bit of the green data section with red (refer to Table 12). Therefore, in MOGCS mode, a RGB input color space will be efficiently divided into four quadrants in a green dimension with red, and a maximum of 2 bits of nine bit addresses will specify the quadrant in the RGB input color section. That is, an output color value is efficiently divided into four quadrants specified with two bit tags. For this reason, the color output value corresponding to each tag value of a certain Rhine will be separated and located in an output color space, and can reduce the aliases of a memory pattern.

[0187] When two bit tags are not in agreement, a data cache control section records a cache mistake, and required memory read-out is started by the data cache control section with cache lookup processing. In addition, a idle state has cache lookup processing until all the values of Rhine corresponding to a 2-bit tag entry are read from external memory and stored in a cache. In this processing, the processing which reads related Rhine of the color translation table stored in external memory is included. Since processing 575 of drawing 63 is performed by each memory bank (for example, 573) of every [of drawing 62], time amount may be needed by the time a result (for example, 586) is outputted from a memory bank depending on the contents of a cache. eight bits sets [32] of data 586 are transmitted to the main data path section (242) after this -- having -- three steps 590-592 of above-mentioned interpolation processing (drawing 62) -- all color channel coincidence -- and it performs by pipeline processing and four ******** 595 sent to a printer device are generated.

[0188] According to the experiment, since the rate of a mistake of the cache in a common image is a cache line fetch for every pixel of 0.01 to 0.03 on an average, it is shown that the cache device shown in drawing 62 and drawing 63 is effective. In many cases, by using such a cache device, the demand to the memory access of the data cache exterior can be reduced sharply.

[0189] Instruction code-ization with two color space conversion modes (drawing 10) which a co-processor performs has the following structures.

Instruction-code[in a color space conversion]-izing [0190]

[Table 12B]

[0191] Drawing 64 shows instruction-field coding in a color space conversion instruction, and my NAOPU

code coding in color conversion command is as follows.
My NAOPU code coding in color conversion command [0192]
[Table 13]
[0193] Drawing 65 shows the technique of changing a RGB pixel stream into a CYMK color value in MOGCS mode. A 24-bit RGB pixel stream is inputted into the pixel organizer 246 (drawing 2) in step S1. At step S2, as drawing 56 and drawing 57 explained, the pixel organizer 246 determines the 4-bit section value of each input pixel, and the location within the 8-bit section using a look-up table. It means in which location within the section the section value of an input pixel and the location within the section exist in which section an input pixel exists again. At step S3, the main data path section 242 combines the red who is an input pixel, green, and the 4-bit section of a blue element, generates a 12-bit address word, and sends this 12-bit address word to the data cache control section 240 (drawing 2). In step S4, as explained in Table 12 and drawing 62, the data cache control section 240 changes this 12-bit address word into eight bit addresses [nine]. These eight addresses show the location in eight memory banks 573 (drawing 62) of output-value valve flow coefficient(P0)-valve flow coefficient (P7). At step S5, the data cache control section 240 (drawing 2) carries out remapping of the eight bit addresses [nine], as drawing 63 explained. Thus, the green maximum bit of the 4-bit section is mapped by a maximum of 2 bits of nine bit addresses with red.
[0194] At step S6, the data cache control section 240 compares a maximum of 2 bits of nine bit addresses with the 2-bit tag in memory 587 (drawing 63). If a 2-bit tag is not in agreement with a maximum of 2 bits of nine bit addresses, output color value valve flow coefficient(P0)-valve flow coefficient (P7) does not exist in cache memory 230. Therefore, in step S7, the output color value corresponding to a 2-bit tag entry is read into a data cache 230 from external memory. In case a 2-bit tag is in agreement with a maximum of 2 bits of nine bit addresses, the data cache control section 240 takes out eight output color value valve flow coefficient(P0)-valve flow coefficients (P7) in the way explained in drawing 62 in step S8. Thus, eight output color value valve flow coefficient(P0)-valve flow coefficients (P7) which enclose an input pixel are incorporated by the main data path section 242 from a data cache 230. At step S7, output color value valve flow coefficient(P0)-valve flow coefficient (P7) is interpolated in the main data path section 242 using the location within the section determined at step S2, and the interpolated output color value is outputted.
[0195] Here, it is clear by dividing further a RGB color space and a corresponding output color value into four or more quadrants, for example, 32 blocks, that the storing field of data cache capacity can be reduced for an expert. When dividing into 32 blocks, the storing capacity of a data cache is good only at 1/32 block of an output color value. Moreover, it is also clear for an expert that the data cache device in which it is used in MOGCS mode can be used in a general single output translation mode. Also in this case, the storing field of a data cache can be reduced.
[0196] 3.17.3 JPEG Coding / Decode
Especially the advantage by encoding and storing an image in saving of memory or the viewpoint of the image transfer rate from a certain location to other locations is unfathomable. Various criteria as image coding currently circulated widely are born. The detailed explanation about a JPEG criterion although one of the very famous criteria is a JPEG criterion is Van. Nostrand Please refer to the prominent book "JPEG:Still Image Data Compression Standard" by Pennebaker and Mitchell which were published by Reinhold in 1993. A co-processor 224 stores an image using the standard subset for JPEG. The standard advantage for JPEG is a point that large compressibility is obtained with image quality maintained. Of course, other criteria may be used in order to compress and store an image. A JPEG criterion is a criterion well known to an expert, and is marketed from the manufacturer in whom the various products which mounted JPEG which can be used for ASICS contain a JPEG core product etc.
[0197] The co-processor 224 is equipped with the function which decodes [which decodes and JPEG-encodes] the image which consists of 1, 3, and 4 color elements. As for 1 color element image, a mesh may not be a mesh, either. That is, either mesh data or the data by which a mesh is not carried out can take out 1 color element. The data with which there are 3 color elements (namely, RGB for every pixel data) for every pixel data as an example of mesh data, and each color element of an image is separately stored, and can process each color element independently as an example of the data by which a mesh is not carried out are mentioned. In the case of 3 color element images, a co-processor 224 assumes that 3 color channels are encoded by a minimum of 3 bytes, and uses 1 pixel for it for every WORD.
[0198] A JPEG criterion divides an image into the small two-dimensional part called the minimum coding part (MCU). Here, each minimum coding part is processed independently. MCU of 16 pixels wide of the image by

which the down sampling was carried out, and 8 pixels long is sufficient as a JPEG encoder (drawing 2), and MCU of 8 pixels wide in the case of the image by which a down sampling is not carried out, and 8 pixels long is sufficient as it.

[0199] Drawing 66 shows the technique of carrying out the down sampling of the 3 element images. Each pixel is stored in the MUV buffer 250 (drawing 2) for the original pixel data 600 by the pixel format to which 601 changes from Y in a YUV color space, U, and V element. This data is changed into the MCU part which consists of four data blocks 601-604 first. A data block is Y element with which the direct sample of the block 601,602 was carried out including various color elements, and blocks 603 and 604 are U and V elements by which the subsample was carried out in the example of drawing 3. Here, a co-processor 224 is equipped with two kinds of subsampling functions. One is a direct sampling which is not filtered, and it leaves odd pixel data and deletes even pixel data. In addition, the average of a neighbor can be taken and U and V element can also be filtered.

[0200] Another JPEG subsampling is 4 color channel subsampling shown in drawing 67. In this subsampling, it has four elements 611 with which the pixel data block of 610 contains 16x8 pixels (O) of opacity elements in addition to the usual Y, U, and V element. A subsample is carried out like [these pixel data 610] drawing 66. However, data blocks 612 and 613 are created in this case using an opaque channel.

[0201] Drawing 68 is drawing which explained the JPEG encoder 241 of drawing 2 to the detail more. JPEG coding / decoder 241 performs the both sides of JPEG coding and decode. Coding processing receives block data from the pixel organizer 246 (drawing 2) through a bus 620. Block data is stored in the MUV buffer 250, and processing is made for every block. JPEG coding processing is divided into some clear steps. These steps are DC performed with the arrangement 4. multiplier encoder 623 of a DCT multiplier with the zigzag scan performed with the quantization 6223. quantizer 622 of the activation 6212.DCT output of the discrete cosine transform in the 1.DCT section. Predicting coding and AC of a DCT multiplier Variable length coding of the output of the multiplier encoder performed with the run length coding 5. Huffman coding vessel 624 of a DCT multiplier. An output is sent to the result organizer 629 (drawing 2) through a multiplexer 625 and Rbus626.

[0202] JPEG decode processing makes JPEG coding actuation reverse. That is, JPEG decode processing includes the processing which inputs the JPEG block compressed from Bus620. compressed data is sent to the Huffman coding machine 624 through Bus630 -- having -- data -- DC -- difference and AC run length decode. Next, it is sent to the multiplier encoder 623, AC and DC multiplier are decoded, and data are returned to the usual scan. Then, reverse quantization of DC multiplier is performed by carrying out the multiplication of the quantization value corresponding to DC multiplier in a quantizer 622. Finally, a reverse discrete cosine transform is given in the DCT section 621, the data of a basis are restored, and it is sent to a result organizer through a multiplexer 625 and Bus626 through Bus631. The JPEG encoder 241 operates by the usual approach through the Standard C bus interface 632 which contains the register set by the instruction control section in order to make actuation of a JPEG encoder start. Moreover, although a quantizer 622 and the Huffman coding machine 624 need a table, this is loaded from a data cache 230 at the time of the need. Table data are accessed through the Obus interface section 634. It connects with the operand organizer B247, and the Obus interface section 634 acts with the data cache control section 240, and suits here.

[0203] The DCT section 621 performs a discrete cosine transform and a reverse discrete cosine transform to pixel data. Although the DCT conversion implementation technique of various classes is known and DCT is described also into "Still Image Data Compression Standard" (same as the above), DCT621 uses the high-speed technique explained in full detail by the following terms "high-speed DCT equipment." In addition, it sets in DCT conversion actuation and is The. Transactions of the IEICE, vol.E71, no.11, November The DCT conversion technique based on the paper "A Fast DCT-SQ Scheme for Images" according to Arai and others for having been carried on 1095 pages of 1988 can also be used.

[0204] A quantizer 622 performs quantization and reverse quantization of a DCT multiplier, and it operates by taking out a related value from the corresponding table stored in the data cache through the Obus interface section 634. In quantization processing, the division of the input data stream is done to it being also at the value read from the quantization table in a data cache. This division is mounted as multiplication of a fixed-point. Moreover, in reverse quantization processing, the multiplication of the data stream is carried out to the value in a reverse quantization table.

[0205] Drawing 69 is drawing which explained the reverse quantization 622 to the detail more. A quantizer 622 is equipped with the DCT interface 640 which passes data to the DCT module 621 or receives data from the DCT module 621 to it through a local bus. In quantization processing, a quantizer 622 receives two DCT

multipliers for every clock cycle. These values are written in one of the internal buffers 641 and 642 of a quantizer. Buffers 641 and 642 are buffers equipped with two ports for carrying out the buffer of the input data. In quantization processing, the multiplier data from the DCT submodule 621 are stored in one of the buffers 641 and 642. If a buffer becomes full, data will be read from a buffer with a zigzag scan, and multiplication will be carried out to it being also at the quantization value received through the Obus interface section 634 with a multiplier 643. This output is transmitted to the multiplier encoder 623 (drawing 68) through the multiplier coding interface 645. While performing these processings, the multiplier of the following block is written in other buffers. In JPEG decode processing, reverse quantization processing is performed by carrying out the multiplication of the DCT multiplier decoded as a quantization module is also at the value stored in the table. In order to carry out actuation with respectively exclusive quantization and reverse quantization, a multiplier 643 is used in the both sides of quantization and reverse quantization. In addition, the location of the multiplier under block of 8x8 is used as an index to a reverse quantization table.

[0206] Like quantization processing, in order that two buffers 641 and 642 may carry out the buffer of the input loading factor data from the multiplier encoder 623 (drawing 68), it is used. The multiplication of the data is carried out to a quantization value, and they are written in a buffer in order of a reverse zigzag scan. If a buffer becomes full, the reverse-quantized multiplier will be read from a buffer to 2 coincidence in the usual sequence, and will be sent to the DCT submodule 621 (drawing 68) through the DCT interface 640. Therefore, the multiplier encoder interface module 645 serves as an interface with a multiplier encoder, and through a local bus, data are sent to an encoder or it reads data from an encoder to it. This module reads data from a buffer in order of a zigzag scan at the time of coding, and writes data in a buffer in order of a reverse zigzag scan at the time of decode. The DCT interface module 640 and the CC interface module 645 can perform read-out and the writing from a buffer. Therefore, it determines using the address / control multiplexer 647 under control of the control module 648 which consists with which buffer each interface is operating of the condition machine for controlling all the modules of a quantizer. A multiplier 643 may carry out the multiplication of the DCT multiplier to a quantization table value using the multiplier of the two's complement of 16x8.

[0207] In drawing 68, the multiplier encoder 623 performs the following functions.

(a) Predicting coding/decode of DC multiplier in JPEG mode

(b) Run length coding / decode of AC multiplier in JPEG mode

In addition, its JPEG mode actuation is desirable, when the multiplier encoder 623 is required and it can be independently used for predicting coding/decode of a pixel, or memory copy actuation. The multiplier encoder 623 performs prediction/run length coding / decode of a DC/AC multiplier as specified in the pink book. Moreover, JPEG which is specified to the JPEG criterion In addition to run length coding / decode of AC multiplier, it also has standard predicting coding / decode function.

[0208] The Huffman coding machine 624 performs the Huffman coding/decode of a JPEG data stream. In Huffman coding mode, the data by which run length coding was carried out from the multiplier encoder 623 are received, and a pack cutting tool's Huffman stream is generated. Moreover, in the Huffman decode mode, the Huffman stream is read from the Pbus interface 620 in a pack cutting tool format, and the multiplier by which the Huffman decode was carried out is sent to the multiplier coding module 623. The Huffman coding machine 624 is stored in a data cache, and the Huffman table accessed through the Obus interface 634 is used for it. Or it can be hard, the Huffman table can be constituted and it can also be made a high speed.

[0209] When using a data cache in Huffman coding, eight banks of a data cache store a data table as explained for every table below at the detail.

Huffman, the quantization table [0210] which are stored in the data cache

[Table 14]

[0211] In drawing 70, the Huffman coding machine 624 mainly consists of two independent blocks with an encoder 660 and a decoder 661. Both blocks 660 and 661 share the same Obus interface through the multiplexer module 662. Each block has an input and an output, respectively and it only becomes active at a point one one of blocking it temporarily according to the function performed with a JPEG encoder.

a. coding in coding JPEG mode -- setting -- the Huffman table -- using -- DC -- difference -- a variable length code can be assigned to a value and AC run length value (to 16 bits per code). The assigned code is sent to HC submodule from CC submodule. Moreover, the Huffman table must be beforehand loaded from the data cache before initiation of operation. And it combines with other bits of DC to which the variable length code has been sent from CC submodule, or AC multiplier, and a pack cutting tool format is generated. Supposing a X'FF cutting tool is obtained as a result of pack processing, X'00 bytes will be inserted. Although a marker is inserted

when a RSTm marker is required, in this case, cutting tool cramming processing in "1" bit of the last Huffman coding and X' 00-byte insertion processing when the stuffed cutting tool becomes X'FF are performed. As for whether it is the need, a RSTm marker is directed with CC submodule. Moreover, HC submodule inserts an EOI marker in the last of an image with directions with the signal of the "last" on a Pbus-CC slave interface. In insertion processing of an EOI marker, the same pack processing as a RSTm marker, cramming processing, and insertion processing are needed. Finally, an output stream is sent to the result organizer 249 as a pack cutting tool, and is written in external memory.

[0212] In the case of non-JPEG mode, data are sent to an encoder as an unpacked data from CC submodule (Pbus-CC slave interface). It encodes independently using the table beforehand loaded to the cache (to JPEG mode and this appearance), and a variable-length symbol is summarized in a pack cutting tool format, and each cutting tool is seen off in the result organizer 249. In addition, as for the cutting tool of the last of an output stream, cramming processing by 1 is performed.

b. Decode

A decode algorithm is equipped with a high-speed (real time) thing and a low-speed thing. A high-speed algorithm operates only in JPEG mode, and a low-speed algorithm operates also in JPEG mode or non-JPEG mode.

[0213] a high-speed JPEG Huffman decode algorithm -- the Huffman symbol -- DC -- difference -- it maps in either a value or AC run length value. Especially this is designed so that it may be suitable for JPEG, and it assumes that the Huffman table (K3, K4, K5, K6) of an example is used at the time of coding. In addition, these tables are embedded in hard into the algorithm so that it can decode without referring to cache memory. Such decode processing assumes a case so that a decode image may be printed, guaranteeing a certain data rate. The data rates of HC submodule which decodes a band (block divided by the RSTm marker) are about one DC / AC multiplier in one clock cycle. Although it may 1 clock-cycle be necessary for [in order to delete an X'00 insertion cutting tool from a data stream] in HC submodule and CC sub inter module, this depends to data strongly.

[0214] The Huffman decoder operates by fast mode and extracts 1 Huffman symbol for every clock cycle. In addition, the high-speed Huffman decoder is described in the following "decoders of a variable-length sign." Moreover, the Huffman decoder 661 is equipped with the low-speed decode algorithm based on a heap, and has the structure 670 shown in drawing 71.

[0215] To a JPEG coding stream, in a stripper 671, an X'00 insertion cutting tool, a X'FF cramming cutting tool, and a RSTm marker are removed, and the Huffman symbol is sent to a shifter 672 with other combined bits. In addition, this processing is not performed in only Huffman's coding stream. The step of the beginning of the Huffman symbol decode is processing which carries out the lookup of the entry of 256 of the HUFVAL table stored in the cache by which addressing was carried out by the first 8 bits of the Huffman data stream. In being the true die length of the Huffman symbol to which this value corresponds, the value concerned is transmitted to the output formatter 676, the symbol length and the number of overhead bits of a decode value are fed back to a shifter 672, a related overhead bit is transmitted to the output formatter 676, and it aligns at least the new initiation section of the Huffman stream sent to the decode section 673. Here, the number of overhead bits is a function of a decode value. When the first lookup does not become a decode value (i.e., when the Huffman symbol is 8 bits or more), heap (located in cache) access is succeedingly performed until the heap address is calculated and it is in agreement, or until "unsuitable Huffman symbol" conditions are fulfilled. When the lookup was in agreement, the same processing as the above is performed and "unsuitable Huffman symbol" conditions are fulfilled, it will be in an interrupt condition.

[0216] The decode algorithm based on a heap is as follows.

It is a loop formation to the last of an image. The symbol length N is set to 8. The first 8 bits of an input stream are stored in INDEX. It is a fetch about HUFVAL (INDEX). If HUFVAL(INDEX) ==00xx000111 .. (ILL) Sending out of an "unsuitable Huffman symbol" signal exit elseif HUFVAL(INDEX) ==1nnn eeee eeee -- (HIT)

It is eeee about a nnn bit. It transmits to eeee as a value. Symbol length N=decimal (nnn) is transmitted. /*000 are */as symbol length 8. Adjustment of an input stream break else/*HUFVAL(INDEX) ==01iii iiii iiii -- (MISS)

HEAPINDEX==ii iiii It sets to iiii (the heap base is assumed to 0).

It sets to N= 9. If The 9th bit of an input stream is 0. It is one increment about HEAPINDEX. fi Fetch of VALUE=HEAP (HEAPINDEX) (bit [9th] sign)

Loop If VALUE==0001 0000 1111--(NL)
Sending out of an "unsuitable Huffman symbol" signal exit elseif VALUE===1000 eeee eeee eeee It is transmitted, using eeee as a value. The symbol length N is transmitted. Adjustment of an input stream break else/*VALUE==01iii iiii iiii -- (MISS)
It sets to N=N +1 (HEAPINDEX=ii iiii iiii).
The Nth bit of an input stream If 0 HEAPINDEX One increment fi The fetch of VALUE=HEAP (HEAPINDEX) pool The pool stripper 671 deletes an X'00 insertion cutting tool, a X'FF cramming cutting tool, and a RSTm marker from an input JPEG671 coding stream, and transmits them to a shifter 672 with the overhead bit with which the "beautiful" Huffman symbol was connected. Since other overhead bits do not exist in only Huffman's coding, the stream transmitted in this mode consists only of the Huffman symbol.
[0217] 672 blocks of shifters are equipped with a 16-bit output register, and they transmit the following Huffman symbol to the decode section 673 (with bit stream of the sequence of MSB to LSB). Determining the bit of which although a symbol is 16 bits or less in many cases, it analyzes is left to the decode section 673. A shifter 672 receives feedback 678, i.e., current symbol length and the overhead bit length following the present (it can set in JPEG mode) symbol, from the decode section 673, and aligns appropriately the initiation time of the following symbol in a shifter 672.
[0218] The decode section 673 mounts the core of the algorithm based on a heap, and is connected to the data cache by the Obus674 course. The decode section 673 is equipped with a data cache fetch block, a lookup value comparator, a symbol length counter, a heap index adder unit, and the decode section (decode is performed based on a decode value) of the number of overhead bits. Here, the fetch address is interpreted as follows.
[0219] Fetch address [0220]
[Table 15]
[0221] The output formatter block 676 performs decode (stand-alone Huffman mode) of a 8-bit value, and association (JPEG mode) to the 32-bit WORD of a 24-bit value, an overhead bit, and RSTm marker information. An overhead bit is transmitted to the output formatter 676 by the shifter 672, after the decode section 673 determines the starting position of the overhead bit to the present symbol. Moreover, the output formatter 673 is equipped with 2 deep FIFO buffer which used 1-word delay in order to predict final value WORD. in decode processing, trying either a high speed and a low speed, if a shifter 672 will decode the cramming bit of the backmost part of an input bit stream arises. Such a condition sends out "forced-termination" signal instead of usually being detected by the shifter and sending out "unsuitable symbol" interrupt by it. if active "forced-termination" signal is sent out, the output formatter 676 sends out 1 latest decode WORD (it still exists in FIFO) as the "last", and does not belong to a decode stream -- the latest WORD is deleted further.
[0222] The detail of the Huffman coding machine 660 in drawing 70 is shown in drawing 72. The Huffman coding machine 660 maps cutting tool data as the Huffman symbol through a look-up table, and is equipped with the look-up table accessed from the coding section 681, a shifter 682, the output formatter 683, and a cache. An input value 685 is encoded in the coding section 681 using the coding table stored in the data cache. Although two tables of the table which should be encoded and which contains a correspondence code for every value, and the table containing code length are needed as a table, in case a symbol is encoded, access to a cache 230 is good at once. In addition, in JPEG compression, another table is needed for every DC multiplier with AC multiplier. Moreover, when subsampling is performed, another table is needed for every non-subsample element with a subsample element. Only two tables (a sign and size) are required of non-JPEG compression. A sign is processed by the shifter 682 and constitutes an output stream from a bit level. Moreover, a shifter 682 also performs EOI marker insertion processing with RSTm which is cutting tool padding processing at the time of the need. And a data byte is transmitted to the output formatter 683, and performs insertion processing by X'00 bytes, cramming processing from a X'FF cutting tool or FF cutting tool before a marker sign, and format processing of a cutting tool by which packing was carried out. In addition, in non-JPEG mode, only format processing of a cutting tool by which packing was carried out is performed.
[0223] Since insertion processing of a X'FF cutting tool is performed by the shifter 682, in order to insert the output formatter 683 before a X'FF cutting tool, the cutting tool of a throat needs to get to know whether you are a marker from a shifter 682. This is performed by having a tag register corresponding to a cutting tool in a shifter 682. Each marker who exists in the byte boundary is tagged by the shifter 682 in marker insertion processing. X"FF" before a marker in the joint processing section 683 -- insertion processing is not performed after a cutting tool. A tag is shifted synchronizing with the main shift register.
[0224] A Huffman coding machine uses two tables for direct Huffman coding using 4 or eight tables in JPEG

compression. The table to be used is shown below.
The table used in a Huffman coding machine [0225]
[Table 16]
[0226] 3.17.4 The table INDEKKUSHINGUHAFU man table is locally stored in the co-processor data cache 230. A data cache 230 is constituted as a direct-mapping cache of 128 lines with which each Rhine consists of 8 words. The address of each WORD in a cache line can be carried out independently, and the Huffman decoder accesses two or more tables at coincidence using this description. Since a table is small (<=256 item), the index to two or more tables in 32 bit-address field of Obus can be included.
[0227] As mentioned above, in JPEG low-speed decode mode, since various Huffman tables are stored, a data cache is used. A format of a data cache is shown below.
The bank address of Huffman / quantization table [0228]
[Table 17]
[0229] Before a JPEG instruction is executed in the JPEG encoder 241 (drawing 2), an image dimension register (PO_IDR) or (RO_IDR) a suitable image width value must be set. With other instructions, an instruction length is related to the number of input data items which should be processed. This relates also to the subsampling option used or the number of color channels also including what kind of padding data.
[0230] All instructions issued by the co-processor 224 use two functions, in order to restrict the amount of output data to generate. These functions are the most effective when the sizes of an input and output data differ, and especially like JPEG coding / decode, when output-data size is strange, they are effective. These functions determine whether to write out output data or only delete data, showing, as the instruction was executed appropriately. By the default, this function is off and it becomes ON by enabling the suitable bit in a RO_CFG register. However, the special option which sets this bit is prepared in the JPEG instruction. In addition, in case JPEG compression is used, as for a co-processor 224, it is desirable to support "deletion" and the "limit" function of output data.
[0231] Deletion and limit processing are explained using drawing 73. The input image 690 has a certain height 691 and a certain width of face 692. Here, some images are interested and the situation of being unrelated to printing other parts often exists. However, a 8x8-pixel block is targetted in a JPEG coding system. Therefore, the case where the width of face of an image does not serve as a multiple of 8, and the case where the field of the Seki core which constitutes MCU695 does not correspond with a boundary exactly arise. Then, output deletion register RO_CUT determines the output byte count deleted in the first part 696 of an output data stream. Moreover, load limitation register RO_LMT determines the maximum output byte count to generate. This maximum output byte count also contains the cutting tool who is not written in memory based on the result of a deletion register. By such processing, the data after the final output cutting tool 698 can ask for a final output cutting tool who is not outputted.
[0232] There are two cases as a case where the deletion in a JPEG decoder and a limit function are especially effective. The 1st case is the case where a part of one strip 701 of a decode image extracts or thaws 700, as shown in drawing 74. The 2nd case is the case where two or more extracts or defrosting of a perfect strip (for example, 711, 712, 713) are needed, in the whole image 714, as shown in drawing 75.
[0233] An instruction format and field coding of a JPEG instruction are shown in drawing 76. Explanation of the my NAOPU code field is described below.
Instruction word-my NAOPU code field [0234]
[Table 18]
[0235] 3.17.5 As for the data coding instruction co-processor 224, it is desirable to have the function in which some JPEG encoders of drawing 2 can be used for other applications. For example, Huffman coding is used also not only in JPEG but in other compression technique. Moreover, it is also desirable to equip the data coding instruction which controls the Huffman coding section only for hierarchical image decode. Furthermore, a run length encoder / decoder, and a predicting-coding machine can also be independently used as the same instruction is also.
[0236] 3.17.6 Perform two-dimensional conversion of a 8x8-pixel block by performing 1-dimensional DCT to the 8x8-block direction of a train first, and subsequently to the line writing direction of a 8x8-pixel block carrying out 1 more-dimensional DCT with discrete cosine transform (DCT) equipment as shown in drawing 77 of the high-speed DCT equipment former. Generally with such equipment, it has an input circuit 1096, an arithmetic circuit 1104, a control circuit 1098, the permutation memory circuit 1090, and an output circuit 1092.

[0237] An input circuit 1096 receives a 8-bit pixel from 8x8 blocks. The input circuit 1096 is connected to the arithmetic circuit 1104 through the middle multiplexers 1100 and 1102. An arithmetic circuit 1104 performs arithmetic processing to 8x8-block a perfect train or a perfect line. A control circuit 1098 controls all other circuits, and performs a DCT algorithm. The output of an arithmetic circuit is sent to the permutation memory 1090, a register 1095, and an output circuit 1092. Permutation memory is further connected to a multiplexer 1100, and a multiplexer 1100 sends out an output to the following multiplexer 1102. Moreover, a multiplexer 1102 also receives the data from a register 1094. The permutation circuit 1090 inputs 8x8 block data in a train format, and outputs data in a line format. An output circuit 1092 outputs the DCT multiplier to 8x8 blocks of pixel data.

[0238] With usual DCT equipment, since the arithmetic circuit 1104 is the most complicated, the rate of an arithmetic circuit 1104 determines the whole equipment rate. The arithmetic circuit 1104 of drawing 77 processes by dividing into two or more processing phases so that arithmetic processing may generally be explained using drawing 78. Therefore, a single circuit which performs each processing phases 1144, 1148, 1152, and 1156 using the usual resources, such as an adder and a multiplier, is used. In such an arithmetic circuit 1104, since it is used in order that a single common circuit may perform the various processing phases of a circuit 1104, it has the fault that a rate becomes slow compared with the optimal rate. Moreover, a storing means to store the intermediate result is also included in this. Since the clock cycle time amount of a circuit must be beyond the at least latest circuit phase, the time amount which the whole processing takes can become more than the sum of the time amount which each processing phase takes.

[0239] Drawing 78 shows the usual arithmetic data path in the equipment of drawing 77, and shows a part of processing which performs DCT in 4 processing phases. In addition, this Fig. is not what showed actual mounting, and shows a function. Each of 4 processing phases 1144, 1148, 1152, and 1156 is built as a circuit in which single reconstruction is possible. Each of 4 processing phases 1144, 1148, 1152, and 1156 of 1-dimensional DCT is reconfigurated for every cycle. Moreover, in this circuit, it is using the pool of common resources (an adder, multiplier, etc.), and each of 4 processing phases 1144, 1148, 1152, and 1156 makes a hardware scale small, and acquires it.

[0240] However, the fault of this circuit is that the rate is not the optimal. As for 4 processing phases 1144, 1148, 1152, and 1156, each consists of same pools of an adder or a multiplier. Therefore, a clock period is determined by the latest processing phase (this example 20ns of block 1144). The whole delay will be set to 27ns if an input, delay (respectively 2ns) of the output multiplexers 1146 and 1154, and delay (3ns) of a flip-flop 1150 are added. Therefore, with this DCT configuration, it operates in 27ns of fastest.

[0241] The DCT configuration of a pipeline format is also known well. The fault of this configuration is a point which needs a lot of hardware. In the viewpoint of a throughput, in the configuration of this invention, although it is less than a pipeline configuration, compared with almost all the current DCT configuration, very good engine performance / size property, and a rate property are shown. Drawing 79 is drawing having shown the configuration of the suitable discrete cosine transform section used in a JPEG encoder (drawing 2) which pixel data are inputted into an input circuit 1126, and stores the train of 8-bit pixel data. Permutation memory changes train formal data into line formal data, in order to carry out the 2nd pass of a two-dimensional discrete cosine transform. In a multiplexer 1124, multiplexing of the memory from an input circuit 1126 and the permutation memory 1118 is carried out, and output data are sent to an arithmetic circuit 1122. The result of an arithmetic circuit 1122 is sent to the output circuit 1120 after termination of the 2nd pass. A control circuit 1116 controls the data flow in discrete cosine transform equipment.

[0242] With the 1st pass of discrete cosine transform processing, the resolution picture multiplier by which inverse transformation is carried out to the string data or pixel data of an image which should be changed is sent to an input circuit 1126. With this pass, a multiplexer 1124 is set up by the control circuit 1116 and data are sent to an arithmetic circuit 1122 from an input circuit 1126. Drawing 80 is drawing having shown the configuration of an arithmetic circuit 1122 in the detail more. In activation of a forward discrete cosine transform, the result of the forward circuit 1138 which performs a forward discrete cosine transform is chosen in a multiplexer 1124. Here, a multiplexer 1124 is set up by the control circuit 1116. In activation of a reverse discrete cosine transform, based on a setup of a control circuit 1126, the output from the reverse circuit 1140 is chosen in a multiplexer 1142. With the 1st pass, after each column vector is processed by the arithmetic circuit 1122 (appropriately set up by the control circuit 1166), the vector concerned is written in the permutation memory 1118. If processing of all 8 column vectors in 8x8 blocks finishes and it is written in the permutation memory 1118, the 2nd pass of a discrete cosine transform will be started.

[0243] With the 2nd pass of a forward or a reverse discrete cosine transform, the vector of a line format is read from the permutation memory 1118, and is sent to an arithmetic circuit 1122 through a multiplexer 1124. With this pass, a multiplexer 1124 disregards the data from an input circuit 1136, and it is set up by the control circuit so that the line vector data from the permutation memory 1118 may be transmitted to an arithmetic circuit 1122. The multiplexer 1142 in an arithmetic circuit 1122 sends data to the output of an arithmetic circuit 1122 the result from the reverse circuit 1140. When the result from an arithmetic circuit 1122 is obtained, based on the command from a control circuit 1116, an output circuit 1120 incorporates a result and outputs it at the subsequent times.

[0244] The arithmetic circuit 1122 is a combinational circuit in that it does not have the storage part which stores the intermediate result. Since the control circuit 1116 grasps the time amount taken to output data through a multiplexer 1124 or an arithmetic circuit 1122 from an input circuit 1136, it can direct correctly the time of incorporating a vector to an output circuit 1120 the result from the output of an arithmetic circuit 1122. While the advantage which does not have middle storage in an arithmetic circuit 1122 can save time amount required for an exchange of the data between middle storage elements, it is mentioned that the time amount taken for data to pass an arithmetic circuit 1122 will serve as the sum of all internal-processing stages, and will not be N times [which requires the greatest time amount] (like [before / of discrete cosine transform equipment]) the processing stage. In addition, N is a processing number of stages in an arithmetic circuit here.

[0245] Drawing 81 shows that the whole delay is only set to the sum of four processing stages 1158, 1160, 1162, and 1164, and 20ns+10ns+12ns+15ns=57ns, and it becomes more nearly high-speed than the circuit of drawing 78. According to such a circuit, the whole system clock cycle can be shortened. In the circuit of drawing 81, supposing four clock cycles are required to obtain a result, it turns out that the minimum execution time is set to 57/4ns (14.25ns) in the whole DCT system, and it will become the large improvement in the engine performance if an example is taken [that a DCT clock cycle cannot but set to 27ns in drawing 78, and].

[0246] It sets at the time of actual activation of this DCT equipment, and is Yukihiro. Arai, Takeshi Agui, Masayuki TheTransactions by Nakajima and others of the The DCT algorithm shown in the paper "the high-speed DCT-SQ technique for an image" carried on page 1095 in IEICE, vol and E71, and no. November, 1988 [11 or] can also be used. By performing this algorithm by hardware, it can arrange easily to the arithmetic circuit 1122 in this DCT equipment. Similarly, it is also possible to arrange other DCT algorithms as hardware in an arithmetic circuit 1122.

[0247] 3.17.7 The example below the Huffman decoder is related with the technique and equipment to the variable-length sign by which the bit field of various die length was interleaved. Especially the example of this invention offers the high-speed decode of a uniprocessing stage (clock cycle) with the sufficient effectiveness of variable-length coded data. Here, in a pretreatment block already different from data which variable length coding is not carried out and have aligned, it shall be deleted from the coding data stream. Furthermore, the positional information of the deleted cutting tool alignment data is sent to the data and coincidence which are decoded at the output of a decoder. Moreover, the cutting tool alignment which remains into the pretreated input data, high-speed detection of a non-variable-length-coding bit field, and a list are provided also with deletion.

[0248] It is desirable to have the high-speed Huffman decoder which can decode JPEG coded data with a rate called 1 Huffman symbol for every clock cycle between marker signs in the suitable example of this invention. In another pretreatment block, cutting tool alignment is carried out from input data, and this separates the marker header by which Huffman coding is not carried out, a marker sign, and an insertion cutting tool, and can realize them by the technique to remove. If the data by which cutting tool alignment was carried out are removed, input data is sent to a data shift combinational circuit block, continuous insertion processing of a data decode register will be performed, and data will be sent to a decode part. A marker's location removed from the input data of a basis is sent to a marker shift block, and the shift of a marker location bit is performed to the input data and coincidence which were shifted in the data shift block.

[0249] The decode section decodes the coding bit field inputted from the data decode register in a combinational circuit. The outputs of the decode section are a decode value (v) and the actual die length (m) of an input sign. Here, m is below n. Moreover, the die length (a) of a variable-length bit field is outputted. Here, a is zero or more values. Since Huffman coding is not carried out, Huffman coding of the variable-length bit field is carried out immediately. The bit field of die-length n under input of the decode section has the die length more than an actual sign. In the decode section, actual code length (m) is determined and it transmits to control block with the die length of other bits (a). Control block determines a shift value (a+m), starts data / marker shift block, and shifts input data in preparation for the following decode cycle.

[0250] With the equipment of this invention, if a decode value, the actual die length of an input sign, and the die length of the bit field by which Huffman coding is not carried out are outputted in predetermined time amount, the decode section of what kind of combinational circuits, such as ROM, RAM, and PLA, can be used. In this example, the decode section outputs a predicting-coding DC multiplier value and AC run length value as prescribed by the JPEG criterion. Moreover, the bit field which was removed from the input data by a decode value and coincidence and by which Huffman coding is not carried out shows the overhead bit which determines the value of DC and AC multiplier as prescribed by the JPEG criterion. As other classification of a bit field which was removed from the data in a data decode register and by which Huffman coding is not carried out, there is a padding bit before the cutting tool alignment marker in the input data stream of a basis as specified to the JPEG criterion. These bits are detected when control block checks the contents of the padding field of a data register. A padding field consists of the k maximum bit of a data register, and is shown by existence of the marker bit in the maximum bit of a marker register. If all the bits in a padding field are the same (a JPEG criterion 1), it is judged as a padding bit, and it will be removed from a data register, without decoding. And the contents of data and the marker register are updated towards the following decode cycle.
[0251] In the example of equipment, it has the output block which performs format processing of output data according to a demand of the suitable example of this invention. An output block outputs a decode value with the signal which shows the corresponding bit field by which variable length coding is not carried out, the input cutting tool who lined up like the marker in JPEG, and the location of a bit field which is not encoded like the overhead bit in JPEG.
[0252] the data decoded with the JPEG encoder 241 (drawing 2) -- JPEG -- it is compatible and it consists of variable-length Huffman-coding codes to which the interleave of the bit field of the fixed length called the bit field which is called an "overhead bit", and by which variable length coding is not carried out, the knitting field which is called the "padding field", and by which variable length coding is not carried out a "marker", a "insertion cutting tool", and a "cramming cutting tool" by which cutting tool alignment be carried out, and which be encoded be carried out Typical input data is shown in drawing 82.
[0253] The configuration and data flow in [whole] the Huffman decoder of the JPEG encoder 241 are shown in drawing 83 and drawing 84. Drawing 83 shows the configuration of the Huffman decoder of JPEG data to the detail. A stripper 1171 removes a marker sign (Signs FFXXhex and XX are non-zero), inserts a cutting tool (sign FFhex), and stuffs a cutting tool (sign 00hex following Sign Ffhex). These are all the elements with which cutting tool alignment of the input data was carried out, and are sent to a stripper as 32-bit WORD. The word [1st] maximum bit which should be processed becomes the head of an input bit stream. In a stripper 1171, the bit field by which cutting tool alignment was carried out is removed from input data, before decode processing of Huffman coding is actually performed in the downstream part of a decoder.
[0254] Input data is inputted into a stripper 1171 as 32-bit WORD for every ** at a clock cycle. Numbering in 3 from 0 is shown for the input cutting tool 1211 in drawing 85. Since the cutting tool of a number (i) is an insertion cutting tool, a cramming cutting tool, or a marker, supposing it is removed, the remaining cutting tools of 0 will be shifted from a number (i-1) to the left with the output of a stripper 1171, and a number (i) will be reduced by one. Under the present circumstances, a cutting tool 0 turns into an "unrelated" cutting tool. A cutting tool's effectiveness outputted from the stripper 1171 is encoded with another output tag 1212 shown in drawing 85. The cutting tool who is not removed by the stripper 1171 is outputted by eye left justification in a stripper. Effective and the tag in which a marker's posterior part is shown are added in that the corresponding cutting tool of each cutting tool under output is effective (a stripper 1171 is passed), and an invalid (removed by the stripper 1171). A tag 1212 controls loading of the marker position to the marker register 1183 through a marker shifter while controlling loading of the data byte to a data register 1182 through a data shifter. Same technique is performed even when 1 bytes or more are deleted from input WORD. That is, all the remaining valid bytes are left justification carried out, and a corresponding output tag shows an output cutting tool's effectiveness. Example 1213 of the output cutting tool to various input cutting tools' combination and an output tag is shown in drawing 85.
[0255] In drawing 83, the role of a pre shifter and the postshifter blocks 1172, 1173, 1180, and 1181 is loading data to a data register and a marker register continuously, when there is sufficient free area for a data register 1182 and the marker register 1183. A data shifter and a marker shifter block are controlled identically [each] and similarly, although it consists of a pre shifter block and a postshifter block. To a data shifter processing the data from a stripper 1171, as for a difference, a marker shifter processes only a tag and is in the point outputted to a decoder at the Huffman value who had the marker position decoded, and coincidence. Transfer direct of the

output of the postshifters 1180 and 1181 is carried out to the registers 1182 and 1183 which correspond to drawing 83 as shown.

[0256] Although the data pre shifter 1172 is shown also in drawing 86, the data pre shifter 1172 adds 32 zero to data from a stripper 1171 at the minimum bit 1251, and extends data to 64 bits. Subsequently, as for extended data, only the number of bits which recognizes current existence at a data register 1182 is shifted by the barrel shifter 1252 of 64-bit width of face on the right. Under the present circumstances, the number of bits is given [the effective bits of which exist in data 1182 and marker 1183 register, and] from the always grasped control logic 1185. And the barrel register 1252 transmits 64 bits to the multiplexer block 1253 which consists of 64 2x1 basic multiplexers 1254. Each 2xbasic 1 multiplexer 1254 considers 1 bit from barrel shifter 1252, and 1 bit from a data register 1182 as an input. When the bit in a data register is effective, a data register bit is outputted. On the other hand, in being invalid, it outputs the bit of barrel shifter 1252. The control signal to all the basic multiplexers 1254 is decoded from shift control 1 signal of control block as shown as a pre shifter control bit 0...5 of the register 1223 in drawing 86 and drawing 87. The output of the basic multiplexer 1254 is sent to barrel shifter 1255, and is shifted to the number-of-bits part left given from the 5-bit control signal shift control 2 as shown in drawing 86. These bits show the number of bits used by decode of the present data in a data register 1182, and if they are the padding number of bits which will be deleted if the present decode Huffman code length, the continuing number of overhead bits, or the padding bit is detected, or below the number of bits from which the number of effective bits in a data register 1182 is deleted, they will become what added 0. Thus, the new data loaded by the data register 1182 after a single decode cycle will be contained in the data outputted from barrel shifter 1255. Since the maximum bit is decoded, the shift-out of the contents of the data register 1182 is carried out from a register, and they are changed into condition that 0, 8, 16, and 24 or 32 bits are added to a data register 1182 from a stripper 1171. When sufficient bit which can be decoded to a data register 1182 does not exist, if the data from a stripper 1171 exist, it is loaded in the present cycle. When the data from a stripper 1171 do not exist in the present cycle, if the decode bit from a data register 1182 is sufficient number of bits, it will be deleted, and if it is not sufficient number of bits, the contents of the data register 1182 will not be changed.

[0257] The marker pre shifter 1173, the postshifter 1181, and the marker register 1183 are the respectively same parts as the data pre shifter 1172, the data post shifter 1180, and a data register 1182. The data flow in parts 1173 and 1181 and 1183 and the data flow between these parts are also the same as that of the data flow between parts 1172 and 1180 and 1182. The same control signal is sent to both part sets from a control section 1185. The difference among these parts is the point how the input data classification of the marker pre shifter 1173 and the data pre shifter 1172 and the contents of the marker register 1183 and the data register 1182 are used. As shown in drawing 88, the tag 1261 from a stripper 1171 is inputted as 8-bit WORD, and is assigned 2 bits for every data byte which faces to a data register 1182. According to the coding technique shown in drawing 85, effective and the maximum bit of a 2-bit tag which shows the cutting tool who is a marker posterior part are 1. Only the maximum bit position of four tags sent to coincidence from a stripper 1171 is sent out as an input 1262 of the marker pre shifter 1173. Thus, the bit to which 1 which shows the location located in a marker's posterior part by the data bit encoded first was set will exist in the input to a marker pre shifter. These have marked the location of the data bit encoded first where a marker follows on coincidence in a data register 1182. By synchronous behavior of the marker location bit in the marker register 1183 and the data bit in a data register 1182, control block 1185 can send out a marker position to the output of a decoder at decode data and coincidence while being able to perform detection and deletion of a padding bit. Since the same control signal is given to two pre shifters (data 1172 and marker 1173), the postshifter (data 1180 and marker 1181), and the register (data 1182 and marker 1183) as above-mentioned, perfect juxtaposition and synchronous operation become possible.

[0258] The decode section 1184 (shown also in drawing 89) is sent to the combinational circuit decode section 1184 for inputting a maximum of 16 bits of a data register 1182, and extracting the decoded Huffman value, the present input code length decoded, and the overhead bit length (it becoming the function of a decode value) following an input sign. Overhead bit length becomes clear when the Huffman symbol before corresponding is decoded, and it becomes the starting position of the following Huffman symbol. Therefore, when maintaining the rate by which one value is decoded for every clock cycle, the Huffman value must be decoded with a combinational circuit block. As shown in drawing 89, as for the decode section, it is desirable to have the decode table of four PLA styles by which inputted the 16-bit token from the data register 1182, and the hard wire was carried out as the Huffman value (8 bits), the corresponding symbol (4 bits) by which Huffman coding

was carried out, and a combinational circuit block which generates an overhead bit (4 bits).
[0259] Deletion of a padding bit is performed in the actual decode processing at the time of a padding bit string being detected in a data register 1182 in the decode section of the padding bit which is a part of control section 1185. The decode section of a padding bit is shown in drawing 90. It is investigated whether a marker location bit exists in the 8 maximum bit of the marker registers 1183 and 1242. When a marker location bit exists, all the bits in the data register 1182 corresponding to the bit before the marker bit in the marker register 1242 and 1241 are judged as a current padding field. As for the contents of the current padding field, it is confirmed by the padding bit detecting element 1243 whether be 1 altogether. When all the bits of the present padding field are 1, it is judged that it is a padding bit and it is deleted from a data register. Here, deletion is performed in shifting the contents of the data registers 1182 and 1241 (they are the marker registers 1183 and 1242 to coincidence) to the left by one clock cycle using the corresponding shifters 1172, 1173, 1180, and 1181. This processing is the same as that of the usual decode mode except for a decode value not being outputted. When all the bits of the present padding field are not 1, not a padding bit deletion cycle but the usual decode cycle is performed. Detection of a padding bit is performed for every cycle as mentioned above, and it is deleted when a padding bit exists in a data register 1182.
[0260] Drawing 87 shows a control section 1185 to a detail. The core of a control section is a register 1223 and holds the currently possessed effect number of bits in a data register 1182. The number of effective bits in the marker register 1183 is always equal to the number of effective bits in a data register 1182. A control section performs three functions. The first function is count of the new number of bits in the data register 1182 stored in a register 1223. The second function is generation of the control signal to shifters 1172, 1173, 1180, 1181, 1186, and 1187, the decode section 1184, and the output-format section 1188. The third function is detection of the padding bit in a data register 1182 as mentioned above.
[0261] In the present number of bits and the present cycle in a data register 1182 (nob), from a stripper 1171, it adds and the new number of bits (new_nob) in a data register 1182 is calculated as what subtracted the number of bits (nor) with the number of bits (nos) in which loading is possible deleted from a data register 1182 in the present cycle. Here, the present cycle is a decode cycle or a padding bit deletion cycle. Therefore, the new number of bits is calculated as follows.
[0262] new_nob=nob+nos-nor -- these processings are performed with an adder 1221 and a subtractor 1222. In addition, (nos) is set to 0 when data are not inputted from a stripper 1171 in the present cycle. Moreover, it is set to 0, also when a bit is insufficient, namely, the bit in a data register is below the sum of the present code length from a control section 1185, and the continuing overhead bit length and decode processing is not performed in the present cycle in a data register 1182 (nos). It is confirmed whether the value (new_nob) might exceed 64 and is over it in the block 1224. In such a case, a stripper 1171 will be in a idle state and loading of new data is not made. A multiplexer 1233 is used in order to make into zero the number of bits loaded from the stripper 1171. Here, the signal which stops a stripper 1171 is not illustrated. The signal "a padding cycle" from the decode section 1231 controls a multiplexer 1234, and chooses it as the number of bits (nor) which should delete the padding number of bits or the decode number of bits (the die length of a sign bit and an overhead bit). If the decode number of bits is judged to be more than the number of bits (nob) in a data register in a comparator 1228, the number of effective bits which is given to a multiplexer 1234 and which should be shifted will be set as zero in NAND gate 1230. That is, (nor) is set as zero and deletion of the bit of a data register is not performed. The output of a multiplexer 1234 is used also for control of the postshifters 1182 and 1183. The width of face of a data register 1182 is set up so that dead lock status may be avoided. That is, it is set up so that only the field in which the maximum number of bits from a stripper 1171 is held may be secured to a data register, or so that the number of effective bits sufficient as a result of decode / padding bit deletion cycle may be deleted.
[0263] Count of the number of bits deleted in a decode cycle is performed in an adder 1226. An operand is inputted from the combinational circuit decode section 1184. Since 16-bit code length is encoded with "0000" in the decode section, by the "ou_reduce" logic 1225, "0000" is encoded by "10000" and an operand without a current sign is obtained. This operand and the output of a subtractor 1227 give the control signal to the output-format shifters 1186 and 1187.
[0264] Block 1229 is used for detection of an EOI (image termination) marker position. Although the EOI marker itself is deleted in a stripper 1171, before deleting by the stripper 1171, the padding bit used as the last bit of the data which existed in the location before an EOI marker exists. In a comparator 1229, it is confirmed whether the number of bits in the data register 1182 stored in the register 1223 is eight or less. With eight [or

less], new data will not be inputted from a stripper 1171 (the remaining bits of the data division by which a data register 1182 is decoded are held), but the padding area size in front of the EOI marker by whom the remaining bits were deleted will be shown. Processing of the further padding field, deletion of a padding bit, etc. are the same as that of the procedure which was used in the case of the padding bit in front of an above-mentioned RST marker.

[0265] Barrel shifters 1186 and 1187 and the output-format section 1188 have **** to support, and can consider various mounting according to an example. Moreover, it may not be mounted at all. The control signal to these is given from a control section 1185 as mentioned above. Only the Huffman-coding length by whom the overhead bit pre shifter 1186 inputs 32 bits, and current decode is done from the data register shifts to the left. Thus, all the overhead bits following the sign by which current decode is carried out will be located in the left to compensate for the output of barrel shifter 1186, and are sent as an input to barrel shifter 1187. The overhead bit post shifter 1187 adjusts an overhead bit location to right alignment from left alignment in 11 bit fields which are used as an output format of data and shown also in drawing 91. An addition bit field is extended to a bit 18 from a bit 8 in the output WORD format 1196, and some of maximum bits also have an invalid thing according to the actual number of overhead bits. This number of bits is encoded by the bits 0-3 of 1196 as prescribed by the JPEG criterion. When using a different format as an output data format, according to a format, barrel shifters 1186 and 1187 and the function of those will be changed.

[0266] The output format block 1188 performs processing which packs a decode value, and performs processing which constitutes in WORD the DC/AC multiplier (1196 bits 0-7) given from a control section 1185, and DC multiplier directions bit (1196 bits 19), the overhead bit (1196 bits 8-18) given from the overhead bit post shifter 1187 and the marker location bit (1196 bits 23) given from the marker register 1183 from a JPEG criterion according to the format shown in drawing 91. The output-format section 1188 also copes with the functional requirement about the output interface of the decode section. Mounting of the output-format section will usually be changed according to it, if an output interface will be changed as a result of a different functional requirement. The above-mentioned Huffman decoder offers very effective decode processing, and realizes high-speed decode processing.

[0267] 3.17.8 an image transformation instruction -- an instruction of these is for performing general affine transformation of a source image. The processing which generates a part of resolution picture is roughly divided into two area. One is the step which determines which part of a source image is connected with the present output scan line, and a step which another performs required subsampling / interpolation processing, and generates an output image for every pixel.

[0268] Drawing 92 shows the flow chart of the step 720 required as that by which the suitable field of a source image is decoded, in order to calculate the purpose pixel value. First, if subsampling is performed, a subsample will be taken into consideration by 721. Next, two processings, such as other interpolation processings 722 and other subsampling processings, are usually mounted. Usually, although interpolation and subsampling are another steps, interpolation and subsampling may be performed together. In interpolation processing, 4 pixels of a perimeter are looked for first, and it determines, before performing congruence linear interpolation 724 for whether the pre multiplication 723 is required. Since computational complexity of congruence linear interpolation processing 724 generally increases very much, thereby, image transformation processing actuation is restrained. The step of the last which calculates the purpose pixel value is processing adding the subsample by which congruence linear interpolation was carried out from the source image. The added pixel value is carried out integral 727 by various approaches, and the purpose image pixel 728 is generated.

[0269] While the instruction word sign for an image transformation instruction is shown in drawing 93, explanation of the my NAOPU code field is shown in the following tables.

Instruction word: My NAOPU code field [0270]

[Table 19]

[0271] Explanation of an instruction operand or result field is shown below.

It is [an instruction operand and] WORD [0272] a result.

[Table 20]

[0273] Operand A points out the data structure known as a "kernel descriptor" which has described all information required in order to define actual conversion. This data structure becomes one of two formats (it defines as L bits of A descriptor). Drawing 94 shows a long sign format of a kernel descriptor, and drawing 95 shows a short sign format. A kernel descriptor describes the following information.

1. Source image initiation coordinate 730 (a fixed length without a sign, 24.24 resolution). A location (0 0) is at

the upper left of an image.
2. Horizontal 731 and Perpendicular 732 (SubSample) Delta (Two's Complement, Fixed Length, 24.24 Resolution)
3. bp field 7334. of triplet which shows location of the binary point in below-mentioned fixed-length matrix multiplier (when it exists) Integral matrix multiplier 735. These are the "adjustable" point resolution (two's complement) of the binary point of 20 which is the location of a binary point implicitly specified by bp field.
5. The rl field 736 which shows remaining numbers of words in kernel descriptor. This value turns into a value which subtracted 1 from what hung the number of trains, and the line count.
[0274] Although the kernel multiplier of a descriptor is put in order for every train, the train which adjoins each other so that it may become a zigzag scan is arranged in hard flow. In drawing 96, Operand B consists of the pointer to the index table which points out the scan line of a source image. The structure of an index table is the structure where an operand B740 points out an index table 741, and an index table points out the scan line (for example, 742) which is a required source image pixel as shown in drawing 96. Generally, a cache is possible for an index table and a source image pixel, and it is located in a local memory.
[0275] Operand C holds the horizontal / perpendicular subsample rate. A horizontal / perpendicular subsample rate is defined by the dimension of the subsample weight matrix specified in case C descriptor exists. The dimension of Matrices r and c is encoded by the data word of an image transformation instruction as shown in drawing 97. The channel N of result pixel P [N] is calculated based on the following formulas.
[0276]
[Equation 4]
[0277] Internally, an integral value is held as a binary point of 36 for every channel. The location of the binary point in the field is specified by BP field. BP field shows the number of bits of the point of the integral result to delete. The integral value of 36 bits is expressed as a two's complement with a sign, and as specified, clamp processing or lap processing is carried out. The example of an interpretation of BP field in a multiplier sign is shown in drawing 98.
[0278] 3.17.9 the reefing processing applied to a reefing instruction rendering image -- a two-dimensional reefing kernel -- a source image -- applying -- a result -- an image -- generating -- a thing -- it is . Reefing processing is usually used in edge radicalization or various image filters. Reefing processing is mounted in a co-processor 224, and is the processing as image transformation processing that a kernel is the same in image transformation processing except the point that 1 source pixel moves for every output pixel by reefing processing to only kernel width of face being moved for every output pixel.
[0279] When a source image has a value S (x y) and a nxm reefing kernel has a value C (x y), the n-th channel of reefing [of S and C] H [n] is [0280].
[Equation 5]
[0281] It is come out and given. Here, they are i** [0, c] and j** [0, r]. The semantics of the semantics of an offset value, the resolution of the intermediate result, and bp field is the same as that of an image transformation instruction. Drawing 99 is drawing having shown the example for which the reefing kernel 750 applies to the source image 751, and generates the result image 752. A source image address generation and output pixel count are performed like an image transformation instruction. An instruction operand is also the same format as image transformation. Drawing 100 shows the instruction word sign of a reefing instruction, and the following tables are explanation of the various fields.
[0282] Instruction word [0283]
[Table 21]
[0284] 3.17.10 Matrix multiplication matrix multiplication is used for color space conversion processing in which the relation of affine transformation exists in two color spaces. Matrix multiplication is defined by the following formulas.
[0285]
[Equation 6]
[0286] WORD has the following formats as a result of a matrix multiplication instruction operand.
It is [an instruction operand and] WORD [0287] a result.
[Table 22]
[0288] Instruction word which shows the my NAOPU code field in the following tables while the instruction word sign for a matrix multiplication instruction is shown in drawing 101 [0289]
[Table 23]

[0290] 3.17.11 The halftone-ized co-processor 224 is equipped with the multiple-value level dither for half toning. The values from 2 to 255 serve as meaningful halftone level. As long as a screen corresponds and it is a mesh or AMMESSHU, either a cutting tool (AMMESSHU or 1 from mesh data) or a pixel (mesh) is OK as the data which carry out a halftone. as [packed / to four output channels (from the same channel to or 4 bytes) / together] -- it is -- it is -- a pack bit (in the case of 2 level halftone) which one sign unpacked for every cutting tool -- or sign (in case of two or more output levels) generation can be carried out.
[0291] An output halftone value is calculated using the following formulas.
(Px(l-1) +d)/255 -- here, p is [the number of level (2<=l<=255) and d of a pixel value (0<=p<=255) and l] dither matrix values (0<=d<=254). The operand sign is as follows.
It is [an instruction operand and] WORD [0292] a result.
[Table 24]
[0293] With an instruction word sign, a my NAOPU code specifies the number of halftone level. An operand B sign is a thing for a halftone screen, and is encoded like tile composition.
3.17.12 Hierarchical Graphics Format Decode
Hierarchical graphics format decode processing contains two or more steps. These steps are the level interpolation, perpendicular interpolation, and Huffman decode and remainder fusion. Each step is performed as another instruction is also. At the Huffman decode step, Huffman coding of the remaining value added to the value interpolated from the interpolation step is carried out. Therefore, the JPEG decode section is used in the Huffman decode.
[0294] Level interpolation processing is shown in drawing 102. An output stream 761 is 764 used as twice as many data as an input stream 672 by which the last data value 763 is reproduced. Drawing 103 is an example which performs 4 times as many level interpolation as this. At the 2nd step of hierarchical graphics format decode, the rise sample of the pixel train is carried out perpendicularly twice or 4 times by linear interpolation. At this step, a 1-pixel train serves as Operand A, and other trains serve as Operand B.
[0295] In perpendicular interpolation, in both case, an output data stream serves as an input stream and a pixel of the same number 4 times twice. The example of the perpendicular interpolation which uses two input data streams 770 and 771, and generates the output stream 772 of 2 double interpolation and the output stream 773 of 4 time interpolation is shown in drawing 104. In pixel interpolation, interpolation processing is performed separately every four channels of four channel pixels.
[0296] Remainder fusion processing includes the addition for every cutting tool of two data streams. The first stream (operand A) is a base value stream, and the second stream (operand B) is a ** value stream. Two input streams 780 and 781 at the time of using remainder fusion processing and the corresponding output stream 782 are shown in drawing 105.
[0297] Drawing 106 shows the instruction word sign of a hierarchical graphics format instruction, and shows the detail of the my NAOPU code field in the following tables.
Instruction word-my NAOPU code field [0298]
[Table 25]
[0299] 3.17.13 an instruction copy instruction -- an instruction of these is divided into two respectively different groups.
a. An instruction of these uses the usual data flow pass in general-purpose data move instruction an organizer 249 and the co-processor 224 which consists of an output interface module as a result of an input interface module, the input interface switch 252, the pixel organizer 246, and the JPEG coding section 241. In this case, a JPEG coding module sends data directly, without processing.
[0300] The following are mentioned as other instructions of data manipulation actuation.
- Packing to the cutting tool of a sub byte value (a bit, a 2-bit value, 4-bit value), packing of the cutting tool within AMPAKKINGU WORD, AMPAKKINGU, alignment and byte lane swapping, and duplicate data manipulation actuation of a duplicate, a memory clearance, and a value are performed in an organizer's (output) combination a result with a pixel organizer (input). In many cases, these instructions are used combining other instructions.
b. Local DMA instruction data manipulation is not performed. As shown in drawing 2, data transfer (both directions) is performed between a local memory 236 and a peripheral interface adapter 237. These instructions are the only instructions with which activation overlaps other instructions. max -- one of the instructions of these "does not overlap" -- it can order simultaneously perform.
[0301] In memory copy actuation, an operand shows the purpose address of a memory copy instruction for the

data which copy Operand A as a result of an example. In a general-purpose memory copy instruction, the data manipulation actuation to an input is prescribed by Operand B, and the actuation to output operand WORD is prescribed by Operand C.
3.17.14 A flow control instruction flow control instruction is an instruction group for controlling various parts of an instruction-execution model as shown in drawing 9. The condition jump or the condition-less jump which performs an instruction stream and enables migration to other addresses from the one virtual address as a flow control instruction at the time of ****** is included. The mask of the field related as a condition jump instruction is also for a co-processor or a register is carried out, and it is determined by comparing with a predetermined value. Thereby, the generality of an instruction can be maintained. Furthermore, a flow control instruction also includes the standby instruction used as a part of microprogramming, in order to take the synchronization between the non-overlapping instructions with an overlap instruction.
[0302] The sign of a flow control instruction is shown in drawing 107. Moreover, the following tables are explanation of a my NAOPU code.
Instruction word-my NAOPU code field [0303]
[Table 26]
[0304] In jump instruction, A words of operands specify the purpose address of jump instruction. If S bits of a my NAOPU code are set to 0, Operand B will specify a co-processor register and will use it as the source of conditions. The value of an operand B descriptor specifies the address of a register, and turns into a value with which the value of B words of operands compares the contents of a register. Operand C WORD specifies the mask for every bit applied to a result. That is, jump instruction conditions will serve as truth, if the formula for every following bits is filled.
[0305] (((register_value xor Operand B)andOperand C)=0x00000000)
Furthermore, the instruction concerned is used also for register access for fully controlling by microprogramming level.
3.18 Explain various modules further in module drawing 2 of an accelerator card.
[0306] 3.18.1 The pixel organizer pixel organizer 246 specifies the address of the data stream from the input interface switch 252, and stores in a buffer. Input data is stored in a pixel organizer's internal memory, or is stored in the MUV buffer 250. After all finishing thing data processing which is the need of receiving an input stream, an input stream is passed to the main data path 242 or the JPEG encoder 241 if needed. The usual CBus interface can constitute a pixel organizer's mode of operation. The pixel organizer 246 operates by one of the five modes which a PO_CFG control register specifies. These modes are as follows.
(a) Idle mode : the mode in which the pixel organizer 246 does not operate.
(b) Sequential mode : it is the mode in which input data comes to be stored in Interior FIFO, the pixel organizer 246 generates 32 bit addresses of data, and data are required of the input interface switch 252.
(c) Color space conversion mode : the mode in which a pixel organizer does the buffer of the pixel for a color space conversion. Furthermore, the interval and fraction which are stored in the MUV buffer 250 are required.
(d) JPEG compress mode : the mode in which the pixel organizer 246 stores image data in a MUV buffer in the form of MCU.
(e) A reefing operation and image transformation mode : the mode which the pixel organizer 246 stores a matrix multiplier in the MUV buffer 250, and will also tell it to the main data path 242 if required.
[0307] The pixel organizer 246 uses the MUV buffer 250 for actuation of both the main data path 242 and the JPEG encoder 241. Setting to a color space conversion, an interval and a fraction table are MUV. It is stored by RAM250 and accessed as 36-bit data (four color channels) x (the interval value of 4 bits, and fraction of 8 bits). Because of image transformation and a reefing operation, it is MUV. RAM250 stores a matrix multiplier and related configuration data. A coefficient matrix is restricted to 16 line x16 train, and the width of face of each multiplier is a maximum of 20 bits. MUVRAM250 needs one multiplier per one clock cycle. In addition to multiplier data, the control information of a binary point and source start coordinate, a subsample delta, etc. must also be told to the main data path 242. The fetch of this control information is done by the pixel organizer 246 ahead of a matrix multiplier.
[0308] In JPEG compression, the pixel organizer 246 does the double buffer of the MCU using the MUV buffer 250. For the improvement in the engine performance of JPEG compression, it is desirable to use a double buffer technique. MUV 1 one half of RAM250 is written in using the data from the input interface switch 252. On the other hand, another one half is read by the pixel organizer in order to obtain the data which should be sent to the JPEG encoder 241. When the size of an input image is not the integral multiple of MCU, the pixel organizer

246 pads MCU, while performing level subsampling of the color component in the place needed.
[0309] The pixel organizer 246 also performs a format of input data including the byte lane swap mentioned above in drawing 32, the normalization, the substitute containing a cutting tool, a cutting tool pack and an unpack, and copy actuation. Actuation is performed if needed by setting up a pixel organizer register. In drawing 108, the pixel organizer 246 is explained more to a detail. The pixel organizer 246 is operating according to control of the register set of the self contained in the CBus interface control 801, and the CBus interface control 801 is connected to the instruction control section 235 via global CBus. The operand fetch section 802 is contained in the pixel organizer 246, and the operand data which the pixel organizer 246 needs is required from the input interface switch 252. The start address of operand data is specified with the PO_SAID register set just before activation. A PO_SAID register may hold immediate data according to assignment by L bits of a PO_DMR register. If a current address pointer is stored in a PO_CDP register and has the demand of an input interface switch, only the burst die length will be increased. Data are MUV. It is MUV as which current offset of data is specified with a PL_MUV register when a fetch is carried out to RAM250. It connects with the base address of RAM250.
[0310] FIFO803 is used in order to carry out the buffer of the sequential input data in which the fetch was carried out by the operand fetch section 802. The data manipulation section 804 performs various actuation which was explained in drawing 32. The output of the data manipulation section is told to the MUV address-generation section 805. The MUV address-generation section 805 follows a configuration register, and is MUV about data. It tells one of RAM250, the main data path 242, and the JPEG encoders 241. The pixel organizer control section 806 is a state machine which generates a control signal required for all the pixel organizer's 246 submodules. In a required signal, the signal which controls the communication link on various Bus interfaces is also included. A pixel organizer control section outputs the diagnostic information which the other modules 239 need according to a setup of a status register.
[0311] In drawing 109, the operand fetch section 802 of drawing 108 is shown more in a detail. The instruction bus address generation section (IAG) 810 is contained in the operand fetch section 802, and the state machine which generates the demand "carry out the fetch of the operand data" is included in it. As for the demand mediation section 811, ****** is arbitrating between the demand of the address-generation section 810, and the demands (drawing 108) of the MUV address-generation section 805 in the demand mediation section 811, and he is trying for this demand to send the demand of a victory to it at the input (MAG) interface switch 252. The demand mediation section 811 contains the state machine for treating a demand. This carries out the monitor of the condition of FIFO using the FIFO count area 814, and determines DESUPATCHI [the next demand] when. The cutting tool enabling generation section 812 generates the cutting tool enabling pattern 816 which specifies the effective cutting tool in each operand with which reception and the input interface switch 252 carry out the return of the information on IAG810. A cutting tool enabling pattern is stored in FIFO with related operand data. When a MAG demand and an IAG demand reach coincidence, the demand mediation section 811 gives priority to a MAG demand over an IAG demand, and processes it.
[0312] In drawing 108, the MUV address-generation section 805 operates in some different modes. In these modes, the 1st is in JPEG (compression) mode. In this mode, the input data for JPEG compression is supplied by the data manipulation section 804, and the MUV buffer 250 is used as a double buffer. MUV The RAM250 address-generation section 805 generates the address of the MUV buffer suitable for storing the input data processed by the data manipulation section 804. MAG805 operates so that 8x8 Blok for JPEG compression may be formed, while generating the read-out address for taking out color component data from the stored pixel. MAG805 is treated also when MCU has lapped with the image in part. Drawing 110 shows an example of the padding actuation which MAG805 performs.
[0313] Setting to ordinary pixel data, MAG805 is MUV of four 8-bit RAM. Four color components are stored in the same address in RAM250. It is MUV after the barrel shift of the MCU data is carried out on the left, in order to take out data from the same color channel to coincidence. It is stored in RAM250. The byte count shifted to the left of data is determined by 2 bits of low order of the write-in address. For example, for drawing 111, when subsampling is not needed, 32-bit pixel data are MUV. The DS arranged within RAM250 is shown. Subsampling of input data may be chosen in three channels or four-channel interleave JPEG mode. It sets to the multi-channel JPEG compress mode accompanied by subsampling, and, for MAG805 (drawing 108), 32 bit data are MUV because of the optimum performance of a JPEG encoder. Subsampling is performed before being stored in RAM250. In four input pixels, it is MUV at first. Data only with useful the 1st and 4th channel which are stored in RAM250 are included. Subsampling of the data of the 2nd and the 3rd channel is carried out, and

they are stored in the pixel organizer's 246 register. In the following four input pixels, the 2nd and the 3rd channel are buried with the data by which subsampling was carried out. Drawing 112 shows an example of the MCU data configuration in multi-channel subsampling mode. MAG treats all single channel unpacked datas completely like multi-channel pixel data. MUV An example of the single channel packed data read from RAM is shown in drawing 113.

[0314] Input MCU is MUV by the write-in process. A read-out process while being stored in RAM is MUV. 8x8 blocks is read from RAM. Generally, said block is generated by every [four multipliers / MAG / 805] by reading data one by one to each channel. In pixel data and unpack input data, the data stored are arranged, as shown in drawing 111. Therefore, in order to compound 8x8 blocks which consists of pixel data by which a sample was not carried out, a read-out process is MUV. It reads carrying out the skew of the data from RAM. Drawing 114 shows an example of such a process. in drawing 114, it can set to four channel data -- read-out SHIKENSU ** is carried out -- having -- **** -- MUV It turns out that it is made easy that the storing format of RAM250 reads many values from the same channel to coincidence.

[0315] It sets to a color translation mode and is MUV. RAM250 is used as a cache which stores an interval and a fraction, and MAG805 commits it as a control section of the cache. MUV RAM250 carries out the cache of the three color channel values. Here, each color channel has 256 pairs of 4-bit intervals, and a fraction. DMU is set to each pixel output which led, and it is MUV. MAG805 is used in order to acquire said value from RAM250. When this value is not acquired, MAG805 advances the memory read-out demand "carry out the fetch of the missing interval and fraction." The technique of fetching many entries is taken instead of the technique of fetching only one entry per demand for a deployment of a band.

[0316] Because of image transformation and a reefing operation, it is MUV. RAM250 has memorized the matrix multiplier of MDP. MAG is MUV. All the matrix multipliers stored in RAM250 are scanned. Earnestly, MAG805 gives a demand to the operand fetch section, and the operand fetch section fetches kernel description "a header" (drawing 94) and the 1st matrix multiplier of a bust demand at the beginning of image transformation and a reefing instruction.

[0317] In drawing 115, the MUV address-generation section (MAG) 805 of drawing 108 is shown more in a detail. MAG805 is equipped with the IBus demand module 820 which multiplexes an IBus demand, and an IBus demand is generated by the image transformation control section (ITX) 821 and the color space conversion (CSC) control section 822. This demand is sent to the operand fetch section which performs a demand. In order that the pixel organizer 246 may operate in one one of the modes of image transformation and a color space conversion, he will not need mediation between control sections 821,822. The IBus demand module 820 derives the information containing the bust address required to generate a demand in the operand fetch section, and bust die length from a related pixel organizer.

[0318] The JPEG control section 824 is equipped with two state machines called a JPEG write-in control section and a JPEG read-out control section, and is used in JPEG mode. Said two control sections take a synchronization to each other by operating to coincidence and using an internal register. In JPEG compression actuation, DMU outputs MCU data, and it is MUV. It stores in RAM. The JPEG write-in control section is taking charge of level padding and control of pixel subsampling, and a JPEG read-out control section takes charge of perpendicular padding. Level padding is performed by suspending a DMU output, and perpendicular padding is performed by reading already read 8x8 blocks again.

[0319] The JPEG write-in control section is carrying out the tracking of the current position of the DCU and the DMU output pixel in a source image, and uses the information for determining when DMU should be stopped for level padding. MCU -- MUV or [that a JPEG write-in control section sets an internal register when written in RAM250] -- or it means whether MCU is in the right edge of an image, or it is in the minimum edge of an image by resetting. It judges whether the JPEG read-out control section was read to MCU of the last of whether perpendicular padding is required and an image based on the contents of said register.

[0320] A JPEG write-in control section carries out the tracking of the DMU output data, and is MUV about DMU output data. It stores in RAM250. Said control section memorizes the current position of an input pixel using a register set. This information is used, when suspending a DMU output and performing level padding. All MCU(s) are MUV. When written in RAM250, said control section writes MCU information in a JPEG-RW-IPC register, and enables it to use it by the JPEG read-out control section henceforth.

[0321] For this control section, the last MCU is MUV. It remains in that condition after being written in RAM250 until it goes into a SLEEP condition and a current instruction is completed. A JPEG read-out control section is MUV. 8x8 blocks is read from MCU stored in RAM250. In a multi-channel pixel, a control section

reads MCU over several times, and it is MUV. A different cutting tool in each read-out is extracted from each pixel stored in RAM.
[0322] It detects whether this control section should perform perpendicular padding using the information offered by JPEG-RW-IPC. Perpendicular padding is MUV. It is carried out by reading again 8 bytes just before reading from RAM250. The image transformation control section 821 reads a kernel descriptor from IBus, and tells a kernel header to MDP242. And only the count specified with the po.len register scans a matrix multiplier. In image transformation and a reefing instruction, the fetch of the data output by PO246 is directly carried out [no] from IBus, and it is told to DMU.
[0323] The 8 bits of the number of the remaining matrix multipliers which should be carried out a fetch are expressed at the beginning of the 1st matrix multiplier to which the immediately after fetch of the kernel header is carried out. Although a kernel header is told to direct MDP, without being corrected, before being told to MDP, the sign escape of the matrix multiplier is carried out. The pixel subsampler 825 is equipped with the two same channel subsamplers with which each operates to 1 byte which is input WORD. When the related configuration register is not started, a pixel subsampler copies an own input to an own output as it is. On the other hand, when the configuration register is started, a subsampler carries out the subsample of the input data by [which take an average to input data or culls out] grazing.
[0324] The MUV multiplexing module 826 chooses MUV read-out and a write-in signal from a control section active now. The internal multiplexing section is MUV. Read-out address output is chosen via the various control sections using RAM250. MUV The RAM write-in address is stored in the 8-bit register of a MUV multiplexing module. MUV The control section using RAM250 is the next MUV. While performing control for determining the RAM address, a write-in address register is loaded.
[0325] For the MUV effective access module 827, the interval and fraction of a current pixel output are used by the color space conversion control section and according to the data manipulation section are MUV. It is determined whether it can use in RAM250. When one or more color channels are missing, the MUV effective access module 827 tells the related address to the IBus demand module 820, and loads an interval and a fraction by the burst mode. If a cache mistake is served, the MUV effective access module 827 will set the internal effective bits showing the set of the interval and fraction by which the fetch was carried out until now.
[0326] Only the count as which an internal pixel register determines a copy module 829 copies input data. It comes to be stopped by the input stream while the copy module has copied current input WORD. The PBus interface module 830 is used the pixel organizer 246, or to carry out the reverse processing. [the main data path 242 and the JPEG encoder 241] Finally, the MAG control section 831 generates the signal which initiates various submodules, and the signal [shut / signal]. In addition, the MAG control section 831 also performs multiplexing to the input PBus signal from the main data path 242 and the JPEG encoder 241.
[0327] 3.18.2 In MUV buffer drawing 2, the pixel organizer 246 is in the MUV buffer 250 and an interrelation so that clearly from old explanation. The MUV buffer 250 in which a reconfiguration is possible is supporting various processing modes containing simple look-up table mode (mode 0), multiplex look-up table mode (mode 1), and JPEG mode (mode 2). The data object of a different type is stored in a buffer in each mode. For example, the value of the data word stored in the buffer and various retrieval tables, single channel data, and multiple channel data are data objects. Generally, a data object has different size. Furthermore, the data object stored in the MUV buffer 250 in which a reconfiguration is possible can actually be accessed by various approaches depending on the operating mode of a buffer.
[0328] In order to return the data of a different type and to make various approaches required to store suitable, a data object is often encoded, before being stored. The approach used for coding of a data object is determined by the size of a data object, a format of the data object currently expressed, and the configuration status of the memory module formed [how a data object is returned from a buffer, and] on the buffer.
[0329] Drawing 116 is the block flow diagram of the component used since the MUV buffer 250 in which a reconfiguration is possible is mounted. The MUV buffer 250 in which a reconfiguration is possible consists of an encoder 1290, a storage device 1293, a decoder 1291, and address reading and a rotation signal generator 1292. When a data object is inputted into the input data stream 1295, an in-house data encodes with an encoder 1290, and a data object is arranged at the internal data stream 1296. The encoded data object is stored in a storage device 1293.
[0330] When decrypting the stored data object, the encoded data are taken out from a storage device by the coded data output stream 1297. The data with which it encoded on the coded data output stream 1297 are decrypted by the decoder 1291. The decrypted data object appears on the output data stream 1298.

[0331] The write-in address 1035 to a storage device 1293 is given by MAG805 (drawing 108). The write-in addresses 1299, 1300, and 1301 are similarly given by MAG805 (drawing 108), and are distributed to a storage device 1293 by address reading and the rotation signal generator 1292. Address reading and the rotation signal generator 1292 generate input / output rotation signals 1303 and 1304 to an encoder and each decoder again. The write-in valid signals 1306 and 1307 are given from the external source. The processing-mode signal 1302 given by the controller 801 (drawing 108) is connected to an encoder 1290, a decoder 1291, address reading and a rotation signal generator 1292, and a storage device 1293. The increment signal 1308 increments the internal counter in address reading and a rotation signal generator, and may be used also in JPEG mode (mode 2).
[0332] When the MUV buffer 250 in which a reconfiguration is possible is in simple look-up table mode (mode 0), a buffer 250 essentially operates like the memory module of a single mode rather. A data object can be picked out from storing or a buffer to a buffer by the approach of essentially accessing a memory module, and the same approach.
[0333] When the MUV buffer 250 in which a reconfiguration is possible is working in multiplex look-up table mode (mode 1), it is in a buffer 250 with a maximum of three retrieval tables stored in the storage device 1293, and is divided into two or more tables. A retrieval table can be accessed simultaneous and independently. When an example is given, as for the table on which the value of an interval and a fraction is stored in the storage device 1293 in multiplex look-up table mode, an index is attached using 3 bytes of low order of the input data stream 1295. 3 bytes of each is published by the independent retrieval table stored in the storage device 1293.
[0334] An image is changed into the encoded data stream when JPEG compression of the image is carried out. A pixel is taken out from a subject-copy image in a format of MCU. MCU is read from the left of an image to the right downward from a top. Each MCU is re-compounded by the block of a large number of 8x8. Much 8x8 blocks are extracted from MCU. MCU is dependent on some factors, such as a color component of a subject-copy image, JPEG mode of a multiple channel, and the need for subsampling. the block of 8x8 -- after that -- Forward DCT (FDCT) -- entropy code modulation is quantized and carried out. In JPEG compression, the encoded data are sequentially read from a data stream. As for a data stream, an entropy decryption, reverse quantization, and reverse DCT (IDCT) are performed. The output of IDCT processing is the block of 8x8. Much 8x8 blocks are unified so that MCU may be reconfigurated. When using JPEG compression, it is dependent on the factor of much above-mentioned [8x8 blocks]. The MUV buffer 250 in which a reconfiguration is possible is used, also when decomposing MCU into much 8x8 blocks or reconfigurating much 8x8 blocks to MCU.
[0335] While the MUV buffer 250 in which a reconfiguration is possible is processing JPEG mode, the input data stream 1295 to a buffer 250 contains the single component which is performing the pixel or JPEG compression processing in which JPEG compression processing is performed. The output data stream of a buffer 250 contains the single channel data block of JPEG expanding processing, or the pixel data of JPEG expanding processing. An input pixel can consist of examples of this JPEG compression to four channels, Y, U, V, and O. When processing processing is carried out as a pixel block which an appointed number of pixels completed, the extract of a single component data block can be started. Each single component data block is constituted by the data which consist of a pixel of this channel stored in the buffer. Therefore, in this example, the single component data block to four can be extracted from one pixel data block. By this example, while the MUV buffer 250 in which a reconfiguration is possible is processing in the JPEG mode for JPEG compression (mode 2), many unit minimum codes (MCU) can store the single of 64, or the pixel of a multiple channel in a buffer, respectively, and can extract it from each MCU in which the component data block of the single channel of many 64-byte length was stored by the buffer. For example, in order that a buffer 1289 may perform JPEG expanding, while being in JPEG mode (mode 2), an output data stream consists of output pixels with a maximum of four components, Y, U, V, and O. When the single component data block which the demanded number completed is written in a buffer, the extract of pixel data can be performed. The cutting tool from four single component data blocks corresponding to the component of a different color is taken out as an output pixel.
[0336] Drawing 117 is the detail drawing of the encoder 1290 of drawing 116. In that of expanding of a pixel block, before each input data object is stored in a storage device 1293, it is encoded by the rotation of the direction of a cutting tool (drawing 129). The magnitude of rotation is determined by the input rotation control signal 1303. In this example, when pixel data are the greatest 4 bytes, the multiplexers 1320 and 1325 with 4 input 1 output of 32 bits are used for selection of the rotation of one out of four possible input pixels. For example, as shown in (3, 2, 1, 0), supposing the label was attached in four cutting tools of a pixel, it will become the rotation (2 (1 (0 (3, 2, 1, 0), 3, 2, 1), 0, 3, 2), 1, 0, 3) of this pixel. Four encoded cutting tools are

outputted to 1290 of a storage device.
[0337] When a buffer is in the modes other than JPEG mode (mode 2), for example, single look-up table mode, (mode 0), and multiplex look-up table mode, the rotation of the direction of a cutting tool is not required, and cannot be performed to an input data object. An input data object receives active jamming by rotation by disregarding the input rotation control signal which has the value of no rotation in the case of the latter. this value 1323 comes out. The multiplexer 1321 of 2 input 1 output generates a control signal 1326 by considering selection of the no operation value 1323 as the input rotation control signal 1303. The current processing mode 1302 is compared with the value in pixel block decomposition mode in order to generate a multiplexer selection signal. . The multiplexer 1320 of 4 input 1 output controlled by the signal 1326 generates the leading data object which chose one of four rotation of an input data object, and was encoded on the encoded input data stream 1326.
[0338] Drawing 118 is a circuit diagram of the combinational circuit which mounts the decoder 1291 which decrypts the encoded output data stream 1297. A decoder 1321 operates by the same approach as an encoder and an essential target. A decoder operates data, only when a data buffer is in JPEG mode (mode 2). 32 bits of low order of the output data object with which it encoded in the data stream 1297 by which the lower part was encoded are passed to a decoder. Data are decrypted using the rotation of the direction of a cutting tool with feeling contrary to carrying out rotation with an encoder 1290. A multiplexer with 4 input 1 output of 32 bits is used in order to choose one of the coded data of four possible classes. For example, as shown in (3, 2, 1, 0), supposing the label is attached in 4 bytes of input pixel, four of the classes (0 (1 (2 (3, 2, 1, 0), 1, 0, 3), 0, 3, 2), 3, 2, 1) of rotation of this pixel are possible. The output rotation control signal 1304 is used when a no operation value is disregarded by other operation modes, when a buffer is a pixel block decomposition node. The no operation value 1333 is 0. The multiplexer 1331 of 2 input 1 output generates a signal 1334 by performing selection of the output rotation control signal 1304 and the no operation value 1333. The current processing mode 1302 is compared with the value in pixel block decomposition mode in order to generate the multiplexer selection signal 1332. The multiplexer 1330 of 4 input 1 output depended and controlled chooses four kinds of rotation of the output data object with which it encoded on the encoded signal 1334** output data stream 1297, and generates output data on the output data stream 1298.
[0339] In drawing 116, the approach of an internal reading address generation used in a circuit is chosen by the processing mode 1302 of the MUV buffer 250 in which a reconfiguration is possible. In single look-up table mode (mode 0) and multiplex look-up table mode (mode 1), it reads and the address is generated by MAG805 (drawing 108) in the form of the external reading addresses 1299, 1300, and 1301. In simple look-up table mode (mode 0), memory modules 1380, 1381, 1382, 1383, 1384, and 1385 (drawing 121) are processed together on a storage device 1293. It reads with the write-in address given to memory modules 1380-1385 (drawing 121), and the address is essentially the same. That is, a storage device 1293 needs only supply of the one reading address and the one write-in address for an external circuit, and in order to distribute these addresses to empty 1385 (drawing 121) from a memory module 1380, it uses internal logic. In the mode 0, the reading address is given by the external address 1299 (drawing 116), and it is distributed to the internal address 1348 (drawing 121), not changing in essence. The external reading addresses 1349, 1350, and 1351 (drawing 121) are not used in the mode 0. The write-in address is given by the external write-in address 1305 (drawing 116), and is essentially connected without correction to the write-in address of each memory modules 1380-1385 (drawing 121).
[0340] Here, the configuration of three look-up tables in multiplex look-up table mode (mode 1) is shown. the input data encoded when three tables were accessed independently -- from 1380 up to 1385 (drawing 121) -- all -- also making a note -- it is written in a joule at coincidence, therefore one index is needed for each of three tables. It is given by the storage device 1293, three indexes, i.e., the reading address, to memory modules 1380-1385 (drawing 212). These reading addresses are distributed to the suitable memory module of 1380 to 1385 using internal logic. The write-in address given from the outside by the same technique as the time of single look-up table mode is essentially connected without essential modification to the address of each memory module of 1308 to 1385. Consequently, in multiplex look-up table mode (mode 1), the external reading addresses 1299, 1300, and 1311 are distributed to the internal reading addresses 1348, 1349, and 1350, respectively. The internal reading address 1352 is not used in the mode 1. The internal address-generation approach used in JPEG mode (mode 2) differs from the above-mentioned approach.
[0341] Drawing 119 is a circuit diagram of the combinational circuit in the JPEG mode (mode 2) in which JPEG compression is performed which mounts the reading address and the rotation signal generation circuit 1292 for the data buffers in which a reconfiguration is possible. In JPEG mode (mode 2), the signal generation

machine 1292 is used in order to calculate the internal reading address of the memory module which includes the output of the component counter 1340 and the data byte counter 1341 for a storage device 1293. The component block counter 1340 generates the component block count which is stored in the storage device and which was extracted from the pixel data block. The block count is given by doubling the output of the data byte counter 1341 four. Specifically, the internal reading addresses 1348, 1349, 1350, and 1351 in pixel block decomposition mode are calculated as follows. It is used in order that it may be used in order that a component block counter may calculate the offset values 1343, 1344, 1345, and 1347, and the output data byte counter 1341 may generate the base reading address 1354. The offset value 1343 is 1358 added to the base reading address 1354, and an aggregate value is the internal reading address 1348 (or 1349, 1350, 1351). Although the offset value of a memory module takes a value different generally to coincidence reading performed with a multiplex memory module, it is essentially the same in the extract of a component block. The same is said of the base address 1354 used for calculating the four internal reading addresses in pixel data block decomposition mode. The increment signal 1308 is used as an increment signal of a component cutting tool counter. Whenever reading is successful, the increment of the counter is carried out. The component block counter increment signal 1356 is used for incrementing the component block counter 1340 after a data block is normally taken out from a buffer in the object for single proofreading.

[0342] The output rotation control signal 1304 (drawing 116) is taken out from the output of a component block counter, and the output of an output data byte counter, and is essentially the same approach as generation of the internal address. It is used for the output of a component block counter calculating the rotation offset 1347. The output rotation control signal 1304 is given by 2 bits of least significants of the sum of the rotation offset 1355 and the base reading address 1354. An input rotation control signal is given like the example of the address and a rotation control signal generation machine by 2 bits of least significants of the external write-in address 1305.

[0343] Drawing 120 is another address-generation machine 1292 used for reconstruction of the multiplexer-channel pixel data from the single component data stored in the MUV buffer 250 in which a reconfiguration is possible. In this case, a buffer serves as JPEG mode for JPEG expanding (mode 2). In this case, a single component data block is stored in a buffer, and a pixel data block is taken out from a buffer. In this example, the write-in address to a memory module is given without essential modification by the external write-in address 1305. A single component block is stored in the continuous memory. The input rotation control signal 1303 of this example is only set by 2 bits of least significants of the write-in address. The pixel counter 1360 is used in order to hold record of the number of pixels extracted from the single component block stored in the buffer. The output of a pixel counter is used in order to generate the reading addresses 1348, 1349, 1350, and 1351 and the output rotation control signal 1304. Generally it reads and the addresses differ for every module which constitutes a storage device 1293. The reading address consists of two parts, the single component block indexes 1362, 1363, 1364, and 1365, or 1365 and a byte index 1361, in this example. In order to calculate the single component block index of a specific block, offset is added to the bits 3 and 4 of an output pixel counter. Generally offset 1366, 1367, 1368, and 1369 differs in each reading address. A bit 0 is used for the byte index 1361 of the reading address from the bit 2 of a pixel counter. The reading address is as a result of association of the single component block indexes 1362, 1363, 1364, and 1365, or a 1365 and a byte index 1361, as shown in drawing 120. The output rotation control signal 1304 is generated without an essential change in this example by the bit 4 and bit 3 of an output of a pixel counter. The increment signal 1308 is used as a pixel counter increment signal for incrementing the pixel counter 1360. When a pixel is normally taken out from a buffer, the increment of the pixel counter 1360 is carried out.

[0344] Drawing 121 is the structure of a storage device 1293. A storage device 1293 can have three 4-bit wide memory modules of 1383, 1384, and 1385, and three 8-bit wide memory modules, 1380, 1381, and 1382. Since the 36-bit WORD in single look-up table mode (mode 0), the WORD of 12x triplet in multiplex look-up table mode (mode 1), the 32 bits pixel in JPEG mode (mode 2), or 4x8-bit single component data is stored, a memory module is combinable. Usually, each memory module is related with the part from which the encoded input and an output data stream (1296 and 1297) differ. For example, a memory module 1380 has the data output port which was connected to the bit 7 from the bit 0 of the encoded input data stream 1296, and was connected with the data input port from the bit 0 of the encoded output data stream 1297 at the bit 7. The write-in address of all memory modules is connected together in this example, and the same value as coincidence is shared. On the other hand, the reading addresses 1386, 1387, 1388, 1390, and 1391 of the memory module shown in drawing 121 are given with the reading address-generation vessel 1292, and these take a value different generally. It is used, in order that a common write-in valid signal may be written in to all 8-bit memory modules and may take

out a valid signal with an example, and the second common write-in valid signal is used in order to write in to all 4-bit memory modules and to take out a valid signal.

[0345] Drawing 122 is a circuit diagram of the combinational circuit for generating the reading addresses 1386, 1387, 1388, 1389, and 1390 for accessing the memory module in a storage device 1293. Each encoded input data object is disassembled into a partial part, and each part is stored in the memory module with which the storage device became independent. Therefore, the write-in address of all the memory modules in all processing modes is essentially the same, and in order to calculate the write-in address of a memory module, logic is usually substantially unnecessary. On the other hand, the reading addresses differ for every processing and usually differ also to each memory module in each processing mode. All the cutting tools in the output data stream 1298 of the MUV buffer 250 in which a reconfiguration is possible have to contain the unit component data extracted from the pixel data stored in the buffer in the JPEG mode (mode 2) JPEG compression, or the pixel data which was stored in the buffer in the JPEG mode JPEG expanding, and was extracted from single component data. The demand to output data is filled by generation of the four reading addresses 1348, 1349, 1350, and 1351 to a buffer. In multiplex look-up table mode (mode 1), since a maximum of three retrieval tables are stored in a buffer, therefore the reading addresses 1348, 1349, and 1350 to a maximum of three attach an index to three retrieval tables, it is required. The reading address of all memory modules is the same as the case in single look-up table mode (mode 0), and only the reading address 248 is used in this mode. The example of the control circuit shown in drawing 122 uses the processing-mode signal of a buffer, and a maximum of the four reading addresses, in order to calculate the reading address 1386-1391 of each of six memory modules which constitute a storage device 1293. The reading address-generation machine 1292 is with the external reading signal which consists of external address buses 1348, 1349, 1350, and 1351 as an input signal, and generates the internal reading addresses 1386, 1387, 1389, and 1390 of the memory module which constitutes a storage device 1293.

[0346] Drawing 123 is drawing having shown how the matrix multiplier of 20 bits would be stored in a buffer 250, when a buffer 250 is in single look-up table mode. In this case, when a data object is written in the MUV buffer in which a reconfiguration is possible, encoding is not usually performed to the data object on a cache. A matrix multiplier is stored in the 8-bit memory modules 1380, 1381, and 1382. A bit 0 is stored in a memory module 1380 from the bit 7 of a matrix multiplier, a bit 8 is stored in a memory module 1381 from a bit 15, and a bit 16 is stored in 4 bits of low order of a memory module 1382 from a bit 19. The data object stored in a buffer which is required because of the remainder of an instruction is taken out repeatedly. Reading of all memory modules and the address of writing in single look-up table mode are essentially the same.

[0347] Drawing 124 is drawing having shown how a table entry would be stored in a buffer in multiplex look-up table mode (mode 1). In this case, three retrieval tables are stored in a buffer and each retrieval table has the interval value of 4 bits, and the fractional value of 8 bits. Usually, an INTABARU value is stored in a 4-bit memory module, and a fractional value is stored in a 8-bit memory module. In this case, three retrieval tables 1410, 1411, and 1412 are stored in memory banks 1380 and 1383, 1381, and 1384, 1382 and 1385. The un-effective control signals 1306 and 1307 (drawing 121) can write an interval value also for the separation past in a storage device 1293, without influencing the fractional value stored in the storage device. A fractional value can be written in without affecting an interval value by the same approach in essence.

[0348] Drawing 125 is drawing having shown how pixel data would be written in the MUV buffer 250 in which the reconfiguration of the condition in the JPEG mode (mode 2) which decomposes a pixel data block into a single element data block is possible. A storage device 1293 is generalized as four 8-bit memory banks which consist of memory modules unified and treated by the same approach as a 8-bit memory module, and memory modules 1380, 1381, 1382, 1383, and 1384 containing 1381 and 1384. A memory module 1385 is not used in JPEG mode (mode 2). It is decomposed into four cutting tools and the encoded 32-bit pixel is stored in the memory module which is 8 bits from which each differs.

[0349] Drawing 126 is drawing having shown how a single component data block would be stored in the storage device 1293 which is in single component mode. A storage device 1293 is generalized as four 8-bit memory banks which consist of memory modules unified and treated by the same approach as a 8-bit memory module, and memory modules 1380, 1381, 1382, 1383, and 1384 containing 1381 and 1384. A memory module 1385 is not used in JPEG mode (mode 2). It is decomposed into four cutting tools and the encoded 32-bit pixel is stored in the memory module which is 8 bits from which each differs. In this case, a single component block consists of 64 bytes. When a single **** component block is written in a ** buffer, the cutting tool rotation of an amount which is different in each is applied. The encoded 32-bit pixel data are taken out by reading the

single component data block from which it differs in a buffer.
[0350] Refer to a pixel organizer's knot for the generalization approach of a data buffer 250 in which a more detailed reconfiguration is possible. By the above example, reconfiguration **** showed that a data buffer was used for the processing of data related to a different instruction. The data buffer with three processing modes in which a reconfiguration is possible was clarified. The generation technique of the different address is needed in each of the processing mode of a buffer. Single look-up table mode (mode 0) is used for storing a matrix multiplier in a buffer in image transformation. It is used for storing many the intervals and fraction retrieval tables in the color space conversion (CSC) of many channels in a buffer in multiplex look-up table mode (mode 1). JPEG mode (mode 2) is used for compounding MCU data to the single component block of 8x8, and re-compounding decomposition or the single component block of 8x8 to MCU in JPEG compression and each JPEG expanding.
[0351] 3.18.3 The result organizer MUV buffer 250 is used also in the result organizer 249. The result organizer 249 does the buffer of the stream of the Maine data path 242 or the JPEG coder 241, and formats it. The result organizer 249 is related also to compression of data, incompressible, denormalization, a byte lane swap, and reorganization again, as a result of drawing 42's explaining. Furthermore, the result organizer 249 transmits the result to the demand of the external-interface controller 238, the local memory controller 236, and the peripheral-interface-adapter controller 237.
[0352] The result organizer 249 is MUV at the time of JPEG expanding mode. RAM250 is used for the image data of the JPEG coder 249 in order to carry out a double buffer. A double buffer can raise the performance, when carrying out JPEG expanding using the data of the JPEG coder 241 currently written in the one half of MUVRAM250 and the image data written in the remaining one half is outputted to coincidence in the appointed storing location.
[0353] 1, 3, and four-channel image data are passed to the result organizer 249 while performing JPEG expanding of the 8x8-block form containing the 8-bit component from the same channel. a result -- an organizer -- these blocks -- assignment -- the mesh of a channel when it stores in MUVRAM250 in sequence and reading is performed for data from MUVRAM250 for the interleave image of a multiple channel is stored. for example, in the JPEG compression of three channels by YUV, the JPEG coder 241 is outputted to introduction Y and a degree by U, and, finally outputs 3 blocks [8x8] in order of V. It is carried out when mesh processing takes out a ********** block or one component, and a pixel consists of forms of (YUVX). X is an intact channel here. Cutting tool swapping is performed when the swap of an output channel is needed. A result organizer also needs to perform required subsampling processing for reconstruction of the chroma data of the elongated output data. This includes the semantics of repeating each program channel, in order to generate.
[0354] If it returns to drawing 127, an organizer's 249 detail is shown as a result of drawing 2. The result organizer 249 has set the foundation on usual the outskirts containing the register file of the register set as the processing of Standard C Bus interface 840. Although processing of the result organizer 249 is the same as that of the pixel organizer 249, reverse data manipulation is performed. The data manipulation unit 842 performs byte lane swapping, component substitution, component release, and denormalization to the data generated with the MUV address generation vessel 805. The performed processing is explained with reference to drawing 42 as above-mentioned, and processing is performed according to various fields set to the internal register. The FIFO queue 843 performs a buffer, before it is outputted using the RBus control unit 844 in output data. The RBus control unit 844 is constituted by an address decoder and the address-generation machine. In addition to the data of a required output byte count, the address for storing modules is stored in an internal register. Furthermore, it is determined whether the internal RO_CUT register was missing before the output cutting tool like how many was seen off on the byte stream of an output bus. in addition, the next data after, as for the RO_LMT register, load limitation was stopped -- using -- max -- it is determined how many data items are outputted. MAG805 generates the address of MUVRAM250 at the time of JPEG expanding. MUVRAM250 is used in order to carry out the double buffer of the output from a JPEG coder. MAG805 performs the mesh of the component in MUVRAM250 depending on an internal configuration register, and performs the output of the single channel containing a pixel, three channels, and four channels. Since byte lane swapping is needed before storing pixel data in a suitable location, the data obtained from MUVRAM250 are passed through a data manipulation unit. When the result organizer 249 is not JPEG mode, MAG805 only sends the PBus receiver's 845 data to the data manipulation unit 842 direct.
[0355] 3.18.4 Returning to the operand organizer B and C drawing 2 again, two independent operand organizers 247 and 248 have the function of the data buffer of the data cache control 240, and the function to transmit data

to the JPEG coder 241 or the Maine data path 242. The operand organizers 247 and 248 are operated in various modes.
(a) Idle mode in which an operand organizer does a chisel response so much at a CBus demand (b) Immediate mode when the data of a current instruction are stored in the internal register of an operand register (c) Sequential mode which generates data when the buffer of the sequential address and the data cache controller 240 is full of an operator organizer.
[0356] The processing mode of many Maine data paths 242 requires one of operand organizers to be sequential mode at least. These modes including the composition in the operand organizer B247 are required of the buffer pixel compounded using other images. The operand organizer C248 is used for the synthetic processing which decreases the value of each data channel. In halftone mode, the operand organizer B247 performs a buffer with a matrix multiplier of 8 bits, and the operand organizer B247 performs the buffer of the data of both perpendicular interpolation and a remainder fusion instruction in hierarchical graphics format decomposition mode.
(d) In stationary mode, the operand organizer B performs repeating the assembly and WORD of single internal data word the number of times specified with the internal register.
(e) In tile mode, the operand organizer B performs the buffer of the data which constitute a pixel tile.
(f) In a random mode, an operand organizer transmits direct the address of MDP242 or the JPEG coder 241 to a data cache controller.
[0357] An internal die-length register determines the number of the items generated by each of the operand organizers 247 and 248 at the time of processing in each mode of sequential one, a tile, and a stationary. the operand organizers 247 and 248 -- each holds the number of the data items processed by ******, and if it reaches the value determined with an internal register, it will stop it. Each operand organizer has reliance in compression, incompressible, and a format [of the input data which used byte lane swapping], substitution [of a component], and normalization function so more. The configuration of the demanded processing is carried out using an internal register. furthermore, the operand organizers 247 and 248 -- in order to restrict a data item, the configuration of each is carried out.
[0358] An operand organizer's (247 248) more detailed configuration is shown by drawing 128. The operand organizers 247 and 248 contain the usual Standard C Bus interface and the register 850 which manages the whole operand organizer's control. Furthermore, it connects with a data cache controller and the OBus control unit 851 controls the data manipulation unit which needs the condition of being saved from the clock cycle of sequential ones, a tile, the address generation in each mode of a stationary, generation of the control signal which enables the communication link of the operand organizer's 247,248 OBus interface, and the past of an input stream and which performs normalization, a repeat, etc. When the operand organizers 247 and 248 are sequential one or tile mode, the OBus controller unit 851 sends the demand of data to a data cache controller. The address is determined by the internal register at this time.
[0359] Each operand organizer contains FIFO buffer 852 of 36-bit width of face used in order to carry out the buffer of the data from the data cache controller 240 in processing in still more various modes. The data manipulation unit 853 performs the same function as the function corresponding to the pixel organizer's 246 data manipulation unit 804.
[0360] The Maine data path / JPEG coder interface 854 distributes the data and the address which are usually exchanged by the Maine data path or the JPEG coder modules 242 and 241 in a processing mode. The MDP/JC interface 854 sends the data from the data manipulation unit 853 to the process constituted so that the Maine data path and its data might be repeated. In the case of a color translation mode, units 851 and 854 are bypassed in order to establish the rapid access of the data cache controller 240 and a color translation table.
[0361] 3.18.5 The description of the example below the main data path section is related with the image processor which offers the computer architecture of the low price which can perform two or more image-processing actuation at high speed. Furthermore, an image processor aims at offering the supple computer architecture which can be constituted so that image-processing actuation which was not ******(ed) from the first may be performed. Moreover, the image processor has much same logic and aims also at offering a computer architecture to which a design process becomes simply and cheap.
[0362] A computer architecture possesses a control register block, a decode block, a data object processor, and flow control logic. A control register block stores all the information about image-processing actuation. A decode block decodes information to a configuration signal, and constitutes an input data object interface. An input data object interface carries out reception storing of the data object from the outside. And these data

objects are distributed to a data object processor. In a certain image-processing actuation, since an input data object interface generates the address of a data object, the source of these data objects can offer a right data object. A data object processor performs arithmetic operation to the received data object. Flow control logic controls the data object flow in data object processing logic.

[0363] Especially a data object processor can be equipped with some same data object subprocessors, and each subprocessor processes some input data objects. A data object subprocessor has some same multifunctional arithmetic sections which perform arithmetic operation to the part concerned of a data object, the after-treatment logic which processes an output data object, and the multiplexing logic which connects the multifunctional arithmetic section and the after-treatment section. The multifunctional arithmetic section possesses the storage for the calculated data object. or [that flow control logic enables this store] -- or DESUEBURU. The multifunctional arithmetic section and multiplexing logic are constituted by the configuration signal generated by decode logic.

[0364] In addition, the configuration signal from decode logic can change with external programming agents. It makes it possible to constitute an image processor so that image-processing actuation which could constitute separately and was not beforehand specified by the external programming agent even if it was what kind of multifunctional block and multiplexing logic may be performed through this mechanism. These descriptions that the example of this invention has, and the other descriptions are explained in full detail below.

[0365] In drawing 2, as mentioned above, the main data path section 242 performs all data manipulation actuation and instructions other than JPEG data coding. Defrosting of composition, a color space conversion, image transformation, a reefing operation, the multiplication of a matrix, half toning, a memory copy, and a hierarchy graphics format is included in these instructions. The main data path 242 sends an output for a pixel and operand data to the result organizer 249 from the pixel organizer 246 and the operand organizers 247 and 248 as a result of reception.

[0366] Drawing 129 is a block diagram of the main data path section 242. The main data path section 242 is a general-purpose image processor, and is equipped with the input interface 1460, an image data processor 1462, the instruction word register 1464, the instruction word decoder 1468, the control signal register 1470, a register file 1472, and ROM1475.

[0367] The instruction control section 235 moves instruction word to the instruction word register 1464 through a bus 1454. Each instruction word includes information, such as a plug which chooses the class of image-processing actuation which should be performed, and various options of image-processing actuation. Instruction word is carried by the instruction word decoder 1468 via a bus 1465. Then, the instruction control section 235 can be instructed to decode instruction word to the instruction word decoder 1468. If the directions are received, the instruction decoder 1468 will decode instruction word to a control signal. And these control signals are carried by the control signal register 1470 via a bus 1469. And the output of a control signal register is connected to the input interface 1460 and an image data processor 1462 via a bus 1471.

[0368] In order to make the main data path section 242 into that more supple, the instruction control section 235 can also write in the control signal register 1470 directly. This can perform [anyone expert in the structure of the main data path section 242] now the fine configuration of the main data path section 242, and the main data path section 242 can also perform now image-processing actuation which is not described by instruction word.

[0369] When all information required in order to perform desired image-processing actuation cannot be held in instruction word, the instruction control section 235 can write all the required information that cannot be held in the register from which some of register files 1472 were chosen. This information is told to the input interface 1460 and an image data processor 1462 via a bus 1473. In a certain image-processing actuation, since the current condition of the main data path section 242 is reflected, the input interface 1460 can update the contents of the register from which the register file 1472 was chosen. When performing image-processing actuation and a problem arises, the instruction control section 235 can discover a problem easily using the above-mentioned description.

[0370] Decode of instruction word is completed, and when the control signal for which it asks to a control signal register is loaded, the instruction control section 235 can be instructed to begin activation of request image-processing actuation in the main data path section 242. If these directions are received, the input interface 1460 will begin to receive the data object from a bus 1451. According to the class of image-processing actuation performed, the input interface 1460 begins to receive the operand data from the operand bus 1452 or the operand bus 1453, or generates the address of operand data and begins to receive the operand data from the operand bus 1452 or the operand bus 1453. According to the output of the control signal register 1470, the input

interface 1460 stores input data and rearranges it. When performing affine image transformation actuation and count like a reefing operation, the input interface 1460 also generates the coordinate by which a fetch should be carried out via buses 1452 and 1453.

[0371] An image data processor 1462 performs the main arithmetic operation to the data object I rehad the input interface 1460 arrange. The image processor 1462 can perform processing which says the underflow of a certain maximum and a data object for the cut-off in various precision over interpolation between two data objects performed by the predetermined interpolation factor, the multiplication of two data objects and the division that divides the result by 255, the usual multiplication to two data objects and addition, and the fraction section of a data object, and overflow of a data object to a certain minimum value as the scaling and clamping of a clamp and a data object which are stopped, respectively. Which under aforementioned arithmetic operation is performed to a data object, or the control signal of a bus 1471 controls the sequence of the actuation etc.

[0372] Although ROM1475 has the dividend of 255/x omitted in 8.8 formats, x is the numbers from 0 to 255 here. ROM1475 is connected to the input interface 1460 and an image data processor 1462 via a bus 1476. ROM1475 generates the blend of short die length, hangs 255 on a data object, and is used for actuation of breaking the result by other data objects.

[0373] Although an operand bus, for example, the number of 1452, is restricted to 2, it is enough in a large majority of image-processing actuation. Drawing 130 shows the input interface 1460 more to a detail. The input interface 1460 equips the data object interface section 1480, the operand interface sections 1482 and 1484, the address-generation condition machine 1486, the blend generation condition machine 1488, the matrix multiplication condition machine 1490, the interpolation condition machine 1494, the data synchronizer 1500, the arithmetic section 1496, the other registers 1498, and a list with the data distribution logic 1505.

[0374] A data object and an operand are thought to be the data object interface section 1480 and the operand interface sections 1482 and 1484 from the exterior. Both of the interface sections 1482 and 1484 are constituted by the control signal from a control bus 1515. The interface sections 1482 and 1484 have the data register containing the data object/operand just received inside, and when two of said data registers are effective, they output a VALID signal. The output of the data register of the interface sections 1482 and 1484 is connected to a data bus 1505. The VALID signal of the interface sections 1482 and 1484 is connected to the flow bus 1510. When it is constituted so that an operand may be fetched, the operand interface sections 1482 and 1484 choose reception and the address required in it for the address from the arithmetic section 1496, the matrix multiplication condition machine 1490, and the output of the data register of the data object interface section 1480 according to the control signal from a control bus 1515. When it is not necessary to store in response to data especially from the exterior in some cases, it is possible that the data register of the operand interface sections 1482 and 1484 is constituted so that data may be stored from the output of the data register of the data object interface section 1480 or the arithmetic section 1496.

[0375] The address-generation condition machine 1486 controls the arithmetic section 1496 in affine image transformation actuation and reefing operation actuation, and calculates the following coordinate by which a source image should be accessed. The address-generation condition machine 1486 waits to set up the START signal of a control bus 1515. A setup of the START signal of a control bus 1515 waits for the address-generation condition machine 1486 to cancel a STALL signal to the data object interface section 1480, and for a data object to arrive. In addition, a counter is set up so that it may become the same as the number of the data objects of the kernel descriptor which needs for the address-generation condition machine 1486 to fetch the address-generation condition machine 1486. The output of a counter is decoded and becomes the enable signal of the data register of the operand interface sections 1482 and 1484, and the other registers 1498. When a VALID signal is started from the data object interface section 1480, the address-generation condition machine 1486 comes to decrease a counter, and is latched to the register with which the next parts of a data object differ.

[0376] If a counter reaches zero, it is directed in the operand interface section 1482 that the address-generation condition machine 1486 should begin to fetch an index table value and a pixel from the operand interface section 1484. In addition, the address-generation condition machine 1486 loads two counters which have the number of lines, and the number of trains, respectively. In all clock edges, when not stopped by the STALL signal from the operand interface section 1482 etc., a counter decreases and outputs the remaining row and columns. And the arithmetic section 1496 calculates the following coordinate by which a fetch should be carried out. When both counters reach zero, a counter loads the number of row and columns again, and the arithmetic section 1496 is constituted so that the upper left edge of the following matrix may be looked for.

[0377] In order to determine the true value of a pixel, when interpolation is used, as for the address-generation

condition machine 1486, the number of a line and trains is decreased every two clock cycles. This is performed using one bit counter and the output is used as enabling [of a line and a train counter]. Once a matrix is scanned, a condition machine sends the signal which decreases the count of a die-length counter. When a counter amounts to 1 and the last index table address is sent to the operand interface section 1482, a condition machine takes out the last signal and resets a start bit.

[0378] The blend generation condition machine 1488 controls the arithmetic section 1496, and generates the sequence of numberses from 0 to 255 for blend die length. This sequence of numbers is used as a interpolation factor which interpolates between a blend starting value and blend exit values. It is decided whether the blend generation condition machine 1488 should be performed in one of the modes (jump mode or step mode). When blend die length is 256 or less, jump mode is used, and step mode is used when that is not right.

[0379] The blend generation condition machine 1488 performs the following count, and sets the result to a register (reg0, reg1, reg2). When a brand lamp is in step mode by the die length determined beforehand, 512-2* die length is latched to reg0 (24 bits), and reg1 (24 bits) and termination-initiation are latched for 511-die length to reg2 (4x9 bits), respectively. 255/(die length -1) is latched to reg0 (24 bits), and it latches reg1 (24 bits) and termination-initiation for 0 to reg2 (4x9 bits), respectively, when a lamp is in jump mode.

[0380] In step mode, the following processings are performed in each cycle. When it is reg 0> 0, reg1 is added to reg0 and the result is stored in reg0. Although it can also enable another incrementer, an output is increased only for 1 in that case. When it is reg 0<=0, 510 is added to reg0 and the result is stored in reg0. It is not increased by the incrementer. The output of an incrementer is a lamp value.

[0381] In jump mode, the following processings are performed in each cycle. reg1 is added to reg0. The output of addition is 24 bits and is outputted in a fixed a small number of point format of 16.8. Said addition output is stored in reg0. Integer part is made to increase when the 1st bit of a fraction result is 1. 8 bits of low order of the integer part of an incrementer are a lamp value. The output of this lamp value 2, i.e., reg, and a blend starting value are sent to an image data processor 1462, and generate a lamp.

[0382] The matrix multiplication condition machine 1490 performs the linearity color space conversion to an input data object using a transformation matrix. A transformation matrix is 4x5 dimensions. Four channels of a data object are hung on the 1st to 4th train, and the backmost row contains the usual state multiplier which should be applied to the sum of a product. When the START signal from a control bus 1515 is started, a matrix multiplication condition machine moves as follows.

[0383] 1) Generate the line number which should carry out the fetch of the usual state multiplier of a transformation matrix from buses 1482 and 1484. In addition, the other registers 1498 are enabled and it enables it to store a usual state multiplier.

2) When it has the 1-bit flip-flop, a line number is generated and the one half of a matrix is fetched from buses 1482 and 1484, use as the address. In addition, the "MAT_SEL" signal which chooses from the one half of a data object what should be hung on the one half of said matrix is also generated.

[0384] 3) When there is no data object inputted from the data object interface section 1480, end.

The interpolation condition machine 1494 performs level interpolation of a data object. In level interpolation, the main data path section 242 interpolates between reception and the next data objects for a data object stream from a bus 1451. And the stream of the data object which is the twice of a former stream or 4 times the die length of this is outputted. Since a data object can be packed to a cutting tool or a pixel, the interpolation condition machine 1494 performs actuation which is different in each case so that a throughput may become max. The interpolation condition machine 1494 operates as follows.

[0385] 1) The data allocation logic 1503 is made to carry out the rearrangement of the input data object, and made to interpolate by generating an INT_SEL signal to a right data object pair.

2) Generate the interpolation factor for interpolating between adjoining data object pairs.

[0386] 3) The data object interface section 1480 generates the STALL signal it is made not to receive a data object any longer. The reason this is needed is that an output stream is longer than an input stream. A STALL signal is sent to the flow bus 1510.

The arithmetic section 1496 possesses the circuit [be / nothing] which performs arithmetic operation, and is constituted by the control signal of a control bus 1515. This is used by only affine image transformation, and a reefing operation and two instructions called the blend generation in composition.

[0387] In affine image transformation and a reefing operation, the arithmetic section 1496 performs the following operations.

1) Calculate following x and a following y-coordinate. In order to calculate an x-coordinate, the arithmetic

section 1496 uses an adder, and adds a horizontal and x components of a perpendicular delta to a current x-coordinate, or lengthens a horizontal and x components of a perpendicular delta from a current x-coordinate using a subtractor. In order to calculate a y-coordinate, the arithmetic section 1498 uses an adder, and adds a horizontal or y component of a perpendicular delta to a current y-coordinate, or lengthens a horizontal or y component of a perpendicular delta from a current y-coordinate using a subtractor.
[0388] 2) Add a y-coordinate to index table offset, and calculate an index table address. In order to calculate the original value of a pixel, when using interpolation, only further 4 is increased in order that the aforementioned sum may ask for an index entry.
3) Add an x-coordinate to an index table entry, and ask for the address of a pixel.
[0389] 4) Subtract 1 from a die-length count.
In blend generation, the arithmetic section 1496 operates as follows.
1) Calculate the internal variable of a lamp generation algorithm in step mode using one certain lamp adder. On the other hand, other one adders are used in order to make a lamp value increase, when an interval variable is larger than zero.
[0390] 2) In jump mode, in order to apply a jump value to a current lamp value, only one adder is needed.
3) A fractional cut-off is performed in jump mode.
4) Lengthen initiation of a brand from termination of a brand in the start of lamp generation.
[0391] 5) Subtract 1 from a die-length count.
The other registers 1498 provide the data object interface section 1480 and a list with excessive storing space other than a data register in the operand interface sections 1482 and 1484. As for the other registers 1498, it is common to be used in storing an internal variable or carrying out the buffer of the data object of the past from the data object interface section 1480. A register 1498 is constituted by the control signal of a control bus 1515.
[0392] The data synchronizer 1500 is constituted by the control signal of a control bus 1515. When [which received the data object in part] a certain interface section does not have other interfaces by providing the data object interface section 1480 and a list with a STALL signal at the operand interface sections 1482 and 1484, the data synchronizer 1500 stops the interface section until it receives other all or data of an interface.
[0393] The data allocation logic 1505 carries out the rearrangement of a data bus 1510 and the data object from a register file 1472 via a bus 1530 according to the control signal of the control bus 1515 including the MAT_SEL signal from the matrix multiplication condition machine 1490, and the INT_SEL signal from the interpolation condition machine 1494. The data by which the rearrangement was carried out are outputted to a bus 1461.
[0394] Drawing 131 shows the image data processor 1462 of drawing 129 more to a detail. An image data processor 1462 has the pipeline control section 1540 and many color channel processors 1545, 1550, 1555, and 1560. All color channel processors receive an input from the bus 1565 driven with the input interface 1460 (drawing 131). All the channel processors and pipeline control sections 1540 are constituted by the control signal from the control signal register 1470 which goes via a bus 1472. All color channel processors may also receive the input from the register file 1472 and ROM1475 of drawing 129 via a bus 1580. It acts as the group of the output of all the color channel processors and pipeline control sections, it serves as a bus 1570, and forms the output 1455 of an image data processor 1462.
[0395] The pipeline control section 1540 controls the flow of the data object of all color channel processors by DESUEBURU [the register of all color channel processors / enable or]. A register pipeline is in the pipeline control section 1540. A pipeline's gestalt and die length are constituted by the control signal from a bus 1471, and that of the gestalt are [the pipeline of the pipeline control section 1540, and the pipeline of a color channel processor] the same. The pipeline control section receives a VALID signal from a bus 1565. On each pipeline stage of the pipeline control section 1540, when an input VALID signal is started and the pipeline stage is not stopped, a pipeline stage latches an input VALID signal while starting a register enable signal to all color channel processors. And it moves to the next pipeline stage, latch's output, i.e., VALID signal. Thus, migration of the data object in a pipeline is simulated and controlled, without using data storage.
[0396] The color channel processors 1545, 1550, 1555, and 1560 perform the main arithmetic actuation to an input data object, and each processor is taking charge of one channel of an output data object. In a suitable example, since a large majority of pixel-data objects have a maximum of four channels, the number of color channel processors is restricted to 4.
[0397] The part which processes the opacity (opacity) channel of a pixel is in a color channel processor. Although not shown in drawing 131, there is a circuit of the addition connected to the control bus 1471, and a

color channel processor changes the control signal from a control bus 1471 so that an opaque channel may be processed correctly. This is because the actuation to an opaque channel differs from the actuation to a color channel for a while in a certain image-processing actuation.

[0398] Drawing 132 shows the color channel processors 1545, 1550, 1555, and 1560 to a detail from (generally 1600 showed drawing 132). Each color channel processors 1545, 1550, 1555, and 1560 are equipped with the processing block A1610, the processing block B1615, the big adder 1620, the fraction cut-off section 1625, a clamp or a wrapper 1630, and the output multiplexing section 1635. The color channel processor 1600 receives [the control signal from the control signal register 1470 / the enable signal from the pipeline control section 1540 / the information from a register file 1472] the data object from the input interface 1460 for the data object from a color channel processor via a bus 1601 via a bus 1603 via a bus 1605 via a bus 1604 via a bus 1602, respectively.

[0399] The processing block A1610 receives the data object from the bus 1601, performs arithmetic actuation of shoes, and outputs the data object calculated partially to a bus 1611. The processing which should be performed for image-processing actuation of the processing block A1610 is explained below. In composition, the processing block A1610 imposes opacity on a data object from the data object bus 1451, interpolates between a blend starting value and blend exit values by the interpolation factor from the input interface 1460 of drawing 129, and carries out the pre multiplication of the operand from the operand bus 1452 of drawing 129, or imposes and grazes opacity to a blend color. And the multiplication to the operand or blend color data by which pre multiplication was carried out is attenuated.

[0400] In a general color space conversion, the processing block A1610 interpolates between four color table values using two fractions from the bus 1451 of drawing 129. In affine image transformation and a reefing operation, the processing block A1610 carries out the pre multiplication of the opacity to the color of a source pixel, and interpolates between the pixels of the same line using the fraction section of a current x-coordinate.

[0401] In a linearity color space conversion, the processing block A1610 carries out the pre multiplication of the opacity to the color of a source pixel, and multiplies the color data by which pre multiplication was carried out by the transformation-matrix multiplier. In level interpolation and perpendicular interpolation, the processing block A1610 interpolates between two data objects.

[0402] In a residual margin, the processing block A1610 adds two data objects. The processing block A1610 is equipped with much multifunctional blocks 1640 and the processing block A GRU logic 1645. The multifunctional block 1640 is constituted by the control signal and can perform one one of the following functions.

[0403] It subtracts and adds to two data objects. One data object is transmitted. Between two data objects is interpolated by a certain interpolation factor. The pre multiplication of the opacity is carried out to a color. Two data objects are hung and the 3rd data object is hung on the product.

[0404] It subtracts and adds to two data objects, and the pre multiplication of the opacity is carried out to the result. DESUEBURU [whether the enable signal from a bus 1604 generated by the pipeline control section 1540 of drawing 131 enables the register of the multifunctional block 1640]. The processing block A GRU logic 1645 is sent to the input of the multifunctional block 1640 with which reception had the data object from a bus 1601 and the data object from a bus 1603, and the output of some multifunctional blocks 1640 chosen, and, as for it, others were chosen in these. The processing block A GRU logic 1645 is also constituted by the control signal from a bus 1602.

[0405] The processing block B1615 performs arithmetic actuation to the data object from a bus 1601, and the data object partially calculated from the bus 1611, and outputs the data object calculated partially to a bus 1616. The processing block B1615 explains below the processing of image-processing actuation performed for accumulating. In composition with a non-forward operator, the processing block B1615 applies the output of ROM which is the value of the 255-/opacity of 8.8 formats to a clamp / data object by which the lap was carried out while imposing the synthetic multiplicand from a bus 1603 to the operand from the pre processed data object and the operand bus 1452 from the data object bus 1451.

[0406] In composition with a forward operator, the processing block B1615 adds two pre processed data objects. Furthermore, in an opaque channel, 255 is subtracted from the aforementioned sum, the difference is imposed on offset, and the product is divided by 255. In a general color space conversion, the processing block B1615 interpolates between four color table values using two fractions from a bus 1451, and interpolates between the color value partially interpolated from the processing block A1610 using the fraction which remains, and former interpolation results.

[0407] In affine image transformation and a reefing operation, using the fraction section of a current y-coordinate, the processing block B1615 interpolates between the pixels interpolated partially, and multiplies the interpolated pixel by the multiplier of a subsample wait matrix. In a linearity color space conversion, the processing block B1615 carries out the pre multiplication of the opacity to the color of a source pixel, and multiplies the color by which pre multiplication was carried out by the transformation-matrix multiplier.
[0408] The processing block B1615 is equipped with much multifunctional blocks and the processing block B GRU logic 1650. Although the multifunctional block is the same as that of the thing of the processing block A1610, in the processing block B GRU logic 1650, it accepts the data object from buses 1601, 1603, 1611, and 1631, and the output of the selected multifunctional block, and sends them to the input of the multifunctional block which had these chosen. The processing block B GRU logic 1650 is also constituted by the control signal from a bus 1602.
[0409] The big adder 1620 combines some of partial results from the processing block A1610 and the processing block B1615. This outputs the result combined with reception and a bus 1621 in a register file 1472 to each input via the processing block B1615 to the bus 1605 via the processing block A1610 to the bus 1616 via a bus 1611 from the input interface 1640 via a bus 1601. The big adder 1620 is also constituted by the control signal of a bus 1602.
[0410] The big adder 1620 can be made a different configuration according to various image-processing actuation. The actuation in predetermined image-processing actuation of the big adder 1620 is explained below. In composition with a non-forward operator, the big adder 1620 adds together two partial products from the processing block B1615.
[0411] In composition with a forward operator, when offset enabling is started, the big adder 1620 lengthens the sum of the data object which has offset from an opacity channel and by which point processing was carried out. In affine image transformation / reefing operation, the big adder 1620 accumulates the product from the processing block B1615.
[0412] In a linearity color space conversion, a big adder adds together two a matrix multiplier / data object products, and usual state multipliers in the 1st cycle. In the 2nd cycle, two the matrix multiplier / data object products which will otherwise accept it are added to the sum of a just before cycle. The fraction cut-off (rounding-off) section 1625 omits reception and the fraction section of an output for the input from the big adder 1620 via a bus 1621. The number showing the fraction section of bits is displayed by BP signal of a bus 1605 from a register file 1472. How to interpret BP signal is expressed to the following tables. A bus 1626 is provided with the omitted output.
[0413] Fraction table [0414]
[Table 27]
[0415] The fraction cut-off section 1625 does two activities in addition to a fractional cut-off.
1) Determine whether the omitted result is negative.
2) Determine whether the absolute value of the omitted result is larger than 255.
Via a bus 1626, reception is followed in an input from the fraction cut-off section 1625, it follows the sequence in the following actuation, and a clamp or a wrapper 1630 is performed.
[0416] the absolute value of the omitted result should be calculated -- ** -- when enabling the option to say, the absolute value is calculated. Overflow of a certain minimum value and a data object is clamped for the underflow of a data object to a certain maximum, respectively. The output multiplexing section 1635 chooses the last output in the output of the processing block B of a bus 1616, the clamp of a bus 1631, or the output of a wrapper. In addition, although some final treatments are also performed to a data object, the following explains the actuation performed for predetermined image-processing actuation.
[0417] In composition with a non-forward operator without pre multiplication, the multiplexing section 1635 combines some outputs of the processing block B1615, and forms a data object without pre multiplication. In composition with a non-forward operator with pre multiplication, the multiplexing section 1635 passes a clamp or the output of a wrapper 1630.
[0418] In composition with a forward operator, the multiplexing section 1635 combines some outputs of the processing block B1630, and forms a data object result. In a general color space conversion, the multiplexing section 1635 applies translation / clamp function to an output data object. In other actuation, the multiplexing section 1635 passes a clamp or the output of a wrapper 1630.
[0419] Drawing 133 shows one multifunctional block like 1640 more to a detail. The multifunctional block 1640 is equipped with the mode detecting element 1710, two addition operand Boolean parts 1660 and 1670,

three multiplexing Boolean parts 1680, 1685, and 1690, 2 input adder units 1675, 2 input multiplication section 1695 with two addends, and a register 1705.
[0420] The mode detecting element 1710 receives the MODE signal 1711 from the control signal register 1470 of drawing 129, and two SUB signals 1712 and the SWAP signal 1713 from the input interface 1460 of drawing 129. The mode detecting element 1710 decodes these signals, and generates the control signal told to addition operand Boolean parts 1660 and 1670 and multiplexing Boolean parts 1680, 1685, and 1690. And this control signal is constituted so that various actuation can perform the multifunctional block 1640. The multifunctional block 1640 has the eight modes.
[0421] 1) Addition-and-subtraction mode : according to the SUB signal 1712, apply an input 1655 to an input 1665, or lengthen from an input 1665. Furthermore, according to the SWAP signal 693, an input is also swappable.
2) Bypass mode : bypass an input 1655 to an output.
3) Interpolation mode : interpolate between inputs 1655 and 1665 by making an input 1675 into a interpolation factor. According to the SWAP signal 1713, inputs 1655 and 1665 are swappable.
[0422] 4) Pre multiplication mode : hang an input 1675 on an input 1655 and divide the result by 255. It teaches the next stage whether the output of the INC register 1708 should increase the result of this stage in a bus 1707, in order to obtain a right result.
5) Multiplication mode : hang an input 1675 on an input 1655.
[0423] 6) Addition and subtraction and pre multiplication mode : an input 1665 is applied to an input 1655, or lengthen from an input 1655, and hang an input 1675 on that result, and divide this product by 255. It teaches the next stage whether the output of the INC register 1708 should increase the result of this stage in a bus 1707, in order to obtain a right result.
[0424] Addition operand Boolean parts 1660 and 1670 ask for the one's complement to an input if needed, in order for subtraction to be also possible with an adder. An adder 1675 adds together the output of the addition operand logic 1660 and 1670 of buses 1662 and 1672, and outputs the sum to a bus 1677. The multiplexing logic 1680, 1685, and 1690 chooses the multiplicand and addend for which it is suitable in order to perform a desired function. These are all constituted by the control signal of the bus 1714 from the mode detecting element 1710.
[0425] The multiplication section 1695 with two addends hangs the input from a bus 1682 on an input from a bus 1677. And the sum of the input from buses 1687 and 1692 is added to said product. An adder 1700 adds 8 bits of high orders of the output of the multiplication section 1695 to 8 bits of low order of the output of the multiplication section 1695. Carry of an adder 1700 is latched to the INC register 1701. A signal 1702 enables the INC register 1701. A register 1705 memorizes the product from the multiplication section 1695. A signal 1702 also enables this.
[0426] Drawing 134 shows the block diagram of synthetic actuation. This synthetic actuation receives three input data streams.
1) Accumulation pixel data : in this accumulation section model, it is guided from the same location as the location where the result was stored.
2) A synthetic operand : it consists of a color and opacity. Both a color and opacity can be a flat, a blend, a pixel, or a tile.
[0427] 3) Attenuation : decrease operand data. Attenuation can be a flat bit map or a cutting tool map.
Pixel data consist of four channels typically. The three channels form the color which is a pixel. The remaining channels are the opacity of a pixel. Pixel data do not have to be carried out even if pre multiplication is carried out. When the pre multiplication of the pixel data is carried out, opacity is imposed on each color channel. Since the formula of synthetic actuation will become easy if the pre multiplication of the pixel is carried out, after the pre multiplication of the pixel data is carried out, usually it is compounded with other pixels.
[0428] The synthetic instruction executed in the suitable example is shown in Table 1. Each instruction works on the data by which pre multiplication was carried out. r means the pixel color ac and opacity a0 to which the pre multiplication of (ac0, a0) was carried out, an "offset" value and wc() mean a lap / clamp operator, and the reverse operator of each operator of over in Table 1, in and out, and atop is also mounted. Moreover, a synthetic model equips left-hand side with an accumulator.
[0429] The synthetic block 1760 in drawing 134 possesses three color subblocks and opaque subblocks. Each color subblock operates to one color channel and opaque channel of an input pixel, and obtains the color of an output pixel. The above actuation is shown below in the form of a pseudo code.

PIXEL Composite(IN colorA,colorB:PIXEL;
IN opacityA,opacityB:PIXEL;
IN comp_op:COMPOSITE_OPERATOR)
(PIXEL result;
IF It is THEN that comp_op is rover, rin, rout, and ratop. colorA and colorB are swapped.;
opacityA and opacityB are swapped.;
END IF;
IF It is THEN that comp_op is over, rover, loado, or plus. X= 1;
ELSE IF It is THEN that comp_op is in, rin, atop, or ratop. X=opacityB;
ELSE IF It is THEN that comp_op is out, rout, or xor. X=not (opacityB);
ELSE IF It is THEN that comp_op is loadzero, loadc, or loadco. X= 0;
END IF;
IF It is THEN that comp_op is over, rover, atop, ratop, or xor. Y=not (opacitya);
ELSE IF It is THEN that comp_op is plus, loadc, or loadco. Y=not (opacitya);
ELSE IF It is THEN that comp_op is plus, loadc, or loadco. Y= 1;
ELSE IF comp_op is in, rin, out, rout, loadzero, or loado. THEN Y= 0;
END IF;
result=coloA * X+colorB * Y;
RETURN result;
Since it has the semantics from which instruction'load' and 'loado' differ to an opaque channel, the above codes differ in an opaque subblock.
[0430] the block 1765 in drawing 134 -- the output of block 1760 -- a clamp -- or a lap is carried out. If it is constituted so that block 1765 may clamp, all larger values than the maximum permitted by the minimum value in all values smaller than the minimum value permitted will be held down to the maximum allowed value. The following formulas will be calculated, if it is constituted so that block 1765 may swap.
[0431] min and max mean the minimum value and maximum which are permitted in a color ((x-min) mod(max-min))+min and here. As the minimum value and maximum, 0 and 255 are desirable. The block 1770 in drawing 134 carries out the pre multiplication of the result from block 1765. This carries out the pre multiplication of the pixel by hanging 255/o on the color value by which pre multiplication was carried out. Here, o means the opacity after composition. The value of 255/o is acquired from ROM in a synthetic engine. The value in ROM is memorized in 8.8 formats, and the part below a fraction is rounded off. The result of multiplication is stored in 16.8 formats. In order to generate the pixel by which reverse pre multiplication was carried out, this result is rounded off by 8 bits.
[0432] The brand generation section 1721 generates the brand of specific die length with a specific starting value and a specific exit value. This is crossed to the following two stages and performed.
1) In lamp generation 2 interpolation lamp generation, a synthetic engine generates the sequence of numbers which carries out linear increase from 0 to 255 to an instruction length. There are [die length] two of 255 or less "jump" "step" modes in lamp generation. [with the mode and die length longer than 255] The mode is decided by 24 bits of high orders of die length. In jump mode, the increment of a lamp value is at least 1 for every clock period. step mode -- it is and the increment of a lamp value is a maximum of 1 for every clock period.
[0433] In jump mode, a synthetic engine uses ROM of 8.8 formats, in order to calculate 255/(die length -1) of step values. This value is applied to a 16-bit accumulator. The output of an accumulator is omitted by 8 bits and forms a sequence of numbers. step mode -- being, a synthetic engine uses the algorithm similar to the drawing algorithm of Bresenham. The algorithm is shown below.
[0434]
Void line draw(length:INTERGER)
{ d=511-length;
incrE=510;
incrNE=512-2*length;
ramp-0;
for(i=0;i(length;i++)
{ if d(=0 then d+=incrE;
else{ d+=incrNE;

ramp++;
}
}
}
Then, the following formula is used in order to generate a brand from a lamp.
[0435]
A cut-off is performed to the division by Blend=(end-start) (x ramp/255)+start255. The above-mentioned formula needs two adders and the block which performs "(as opposed to end-start) pre multiplication" with the lamp of each channel. Other image processings which the main data path section 242 can perform are general color space conversions. A generalization color space conversion (GCSC) uses piece wye ZUTORAI linear (3rd linearity) interpolation, in order to calculate an output color value. It is desirable to perform conversion to-dimensional [1] or 4-dimensional output space of a three dimension from input space.
[0436] In some cases, the accuracy of the TORAI linear interpolation in the edge of a color gamut becomes a problem. This problem becomes remarkable in a sensitive printing device to near an edge. in order to avoid this problem, GCSC is alternatively calculated in an extended output color space -- it can have -- the following formula -- using -- suitable within the limits -- a scale -- and it is clamped.
[0437]
They are image transformation and a reefing operation at the image processing of others which can perform a suitable example. image transformation -- setting -- a source image -- a scale -- a skew is rotated and carried out. In a reefing operation, the pixel of a source image is collapsed, is sampled with a matrix, and generates the purpose image. The next phase is required in order to generate the scan line in the purpose image.
[0438] 1) Carry out inverse transformation of the scan line of the purpose image as shown in drawing 135. The pixel of a source image required for this to generate the scan line of the purpose image is discriminable.
2) Thaw the need part of a source image.
3) Carry out inverse transformation of the horizontal of the purpose image, perpendicular subsampling distance, Initiation x, and the y-coordinate to a source image.
[0439] 4) Transmit the above-mentioned information to the processing section, perform required subsampling and interpolation, and ask for the pixel of an output image.
The writing of subsampling, interpolation, and the purpose pixel etc. is performed by the suitable example, and count of the related part in a source image, the subsampling frequency which should be used is performed by host application.
[0440] Drawing 136 is a block diagram of a required phase in count of the purpose pixel value. Drawing 136 is assumed to be what has the available pixel of a required source image. The phase of the last which calculates the purpose pixel is adding together all the subsamples by which secondary linear interpolation's was carried out from the source image. The block diagram of the image transformation engine pulled out by suitable setup in the main data path section 242 is shown in drawing 137. The image transformation engine 1830 consists of Boolean part 1835 which calculates the address-generation section 1831, the pre multiplication section 1832, the interpolation section 1833, the accumulation section 1834, omission, a clamp, and an absolute value.
[0441] The address-generation section 1831 generates x of a source image required to constitute a result pixel, and the y-axis. Moreover, this generates the address for asking for an index offset from the pixel of the input index table 1815 and an image 1810. A kernel descriptor is read before the address-generation section 1831 generates x of a source image, and the y-axis. There are two classes of formats of a kernel descriptor, and it is shown in drawing 138. A kernel descriptor is the initiation coordinate (a fixed-point without a sign, 24.24 precision) of 1 source image. A location (0 0) is the upper left edge of an image.
[0442] 2) A horizontal, a perpendicular subsample delta (a two's complement, 24.24 precision)
3) bp field of the triplet which shows the location of the binary point in a fixed-point matrix multiplier. Drawing 150 shows a definition and its explanation of bp field.
4) Accumulation matrix multiplier. This is the thing of "adjustable" decimal point precision with the binary location (two's complement) of 20 pieces, and the location of the binary point is implicitly specified by bp field.

[0443] 5) rl field which shows the remaining number of the WORD of a kernel descriptor. This value is the same as what hung the number (number -1 of a train) of a line.
In a short kernel descriptor, other parameters except the constant section of the initiation coordinate of x have the following values.

fraction of the initiation coordinate of x < -- -0 and initiation coordinate of y < -- -0 and level delta < -- -1.0 and perpendicular delta < -- -1.0
The present coordinate is calculated after the address-generation section 1831 is constituted. There are two approaches in this according to the dimension of a subsample matrix. When the dimension of a subsample matrix is 1x1, the address-generation section 1831 adds a level delta to the present coordinate until sufficient coordinate is acquired.
[0444] When the dimension of a subsample matrix is not 1x1, the address-generation section 1831 adds a level delta to the present coordinate until one line of a matrix finishes. Then, the address-generation section 1831 adds a perpendicular delta to the present coordinate, in order to search for the coordinate of the following line. The address-generation section 1831 lengthens a level delta from the present coordinate until one or more trains finish, in order to search for the following coordinate. Then, the address-generation section 1831 adds a perpendicular delta to the present coordinate, and repeats this process. The diagram in the upper limit of drawing 150 shows the access approach to a matrix. Since a matrix is zigzag, it is scanned and current x and the y-axis are calculated by this approach using this structure, the required number of registers is good at least. A accumulation matrix multiplier must be put in order in the same sequence in a kernel descriptor.
[0445] After generating the present coordinate, the address-generation section 1831 adds the y-axis to an index table base address in order to ask for the address of an index table (when the source pixel is interpolated, the address-generation section 1831 needs to ask also for the following index table). An index table base address points out the index table entry in (y+0). After asking for an index offset from an index table, the address-generation section 1831 adds it to an x-coordinate. This sum is used when asking for 1 pixel from a source image (it is 2 pixels when the source pixel is interpolated). When the source pixel is interpolated, the address-generation section 1831 adds an x-coordinate to the following index offset, and obtains two or more pixels.
[0446] When searching for the coordinate of image transformation, similar technique is used also in a reefing operation. The only difference with a reefing operation is that, as for a reefing operation, the initiation coordinate of the matrix in the following output pixel has separated only the level delta from the initiation coordinate of the matrix in a front pixel. In image transformation, the initiation coordinate of the matrix in the following pixel has separated only the level delta from the coordinate of the top right corner pixel of the matrix in a former output pixel.
[0447] In drawing 139, the diagram of the middle shows the above-mentioned difference. If the pre multiplication section 1832 is required, it will hang the color channel and opaque channel of a pixel. The interpolation section 1832 interpolates a source pixel in order to ask for the true color of a required pixel. This takes 2 pixels from source image memory, interpolates them using the fraction part of a current x-coordinate, and inputs the result into a register. Then, 2 pixels of the next train of source image memory are taken, and, similarly it interpolates using the fraction of x. Then, the interpolation section 1833 interpolates this interpolation value and the interpolation value before that using the fraction section of a current y-coordinate.
[0448] The accumulation section 1834 does two activities.
1) Spend a matrix multiplier and a pixel.
2) Output the value which accumulated the result when all matrices are received to the next stage. The initial value of the accumulation section 1834 is initialized by 0 or the specific value according to a channel.
[0449] Block 1835 omits the output of the accumulation section 1834, and if required, it will restrict an underflow and the overflowing value to maximum or the minimum value. And if required, the absolute value of an output may be calculated. The location of the binary point is specified by bp field of a kernel descriptor in the output of the accumulation section. bp field shows the number of the bits which should be thrown away in a accumulation result. This is shown in the diagram of the lower limit in drawing 139. This accumulation value is treated as a two's complement with a sign.
[0450] Another image-processing actuation which can perform the main data path section 242 is matrix multiplication. Matrix multiplication is used for a color space conversion in case there is affine relation between two space. This is a difference with a general (based on 3rd linear interpolation) color space conversion. The result of matrix multiplication is defined by the following formula.
[0451]
[Equation 7]
[0452] Here, ri is a result pixel and ai is A operand pixel. The sizes of a matrix must be five trains of four lines. Drawing 140 is a block diagram of the multiplication-adder which performs matrix multiplication in the main data path section 242. It consists of Boolean part which calculates an absolute value by clamping an output

value the multiplication section which applies a matrix multiplier into this at a pixel channel, the adder which adds that result together, and if needed.
[0453] In order to complete matrix multiplication, two clock cycles are required. The multiplexing section is set up for every cycle and the data of the multiplication section and an adder unit are chosen correctly. In the 0th cycle, 2 bytes of least significant of a pixel is chosen by the multiplexing sections 1851 and 1852. Next, two trains in the left-hand side of a matrix, i.e., the matrix multiplier in the 0th line in a cache, are multiplied by the multiplier.
[0454] 2 bytes of high order is chosen from that of a pixel by the top multiplexing section in the 1st cycle. Next, two trains in the right-hand side of a matrix are multiplied by the multiplier. The result of multiplication is 1854 added to the result of the last cycle. The sum in an adder unit is 1855 omitted by 8 bits. "Operand Boolean part" 1856 carry out the rearrangement of the multiplication section output so that the input of an adder unit 1854 may be set to four. This performs the rearrangement for enabling addition to the result of the multiplication section, and it is made to output the right product of a 24-bit multiplier and a 8-bit pixel component.
[0455] "AC Boolean part" 1855 calculate the absolute value of the result omitted according to omission and a setup in 12 bits of least significants of the output of an adder unit. then, a setup -- responding -- the result -- a clamp -- or a lap is carried out. When it is set up so that "AC Boolean part" may clamp, all zero or less values are held down by 0, and all 255 or more values are held down by 255. When it is set up so that a lap may be carried out, 8 bits of low order of a constant part are outputted for "AC Boolean part."
[0456] The main data path section 242 can also be set up so that image processings other than the above may be performed. While cost is reduced by design reuse, the computer architecture which can perform various image-processing actuation early is described below. In addition, if it is used even to that architecture even if it is an external programming agent since this computer architecture has flexibility, it can constitute a computer so that image-processing actuation which was not predicted from the first can also be performed. Moreover, since the core of a design mainly consists of some multifunctional blocks, it can reduce difficulties of a design remarkably.
[0457] 3.18.6 The data cache control section and the cache data cache control section 240 are equipped with 4 K bytes of read-out data cache 230 in a co-processor 224. The data cache 230 is arranged as a direct map RAM cache, and direct mapping can be carried out to Rhine where the same die length in cache memory 230 (drawing 2) is the same as for all of Rhine with the same die length in external memory. A cache line, a call, and the above-mentioned cache memory usually consist this Rhine in cache memory of a majority of such cache lines.
[0458] The data cache control section 240 serves the data demand from two operand organizers 247 and 248. It checks whether data exist in a cache 230 first. Otherwise, the fetch of the data is carried out from external memory. There is the address-generation section which can do a program in the data cache control section 240, and the data cache control section 240 makes it possible to operate in the addressing mode from which some differ. Moreover, the special addressing mode which comes to be made by the data cache control section 240 also has the address of the demanded data. In this mode, the data to 8 words (256 bits) can be sent to the operation organizers 247 and 248 at coincidence.
[0459] Cache RAM -- eight -- it consists independently of a memory bank in which the address is possible (the address was carried out by the different Rhine address). The data from each bank are required for the special addressing mode of eclipse ****** with a unit to 256 bits. This arrangement can serve even eight 32-bit demands for coincidence, if it comes from a bank which is different in each other.
[0460] A cache operates in the following modes later mentioned in a detail. If required, it is also possible to make it automatically put into all caches. 1. -- general normal mode 2. single output color space conversion mode 3. -- many -- general output color space conversion mode 4.JPEG coding mode 5. low-speed JPEG decode mode 6. matrix multiplication -- mode 7. DESUE bull mode 8. nullification mode Fig. 141 shows the address, the data, the flows of control, and the data cache 230 of the data cache control section 240 in drawing 2.
[0461] A data cache 230 possesses the direct map cache mentioned above. The data cache control section 240 possesses the tag memory 1872 which has a tag entry in each cache line, and a tag entry has the top section of the external memory address to which the current map of the cache line is carried out. Moreover, it also has the Rhine effective condition memory 1873 which shows whether a current cache line is effective. The initial state of all cache lines is invalid.
[0462] The data cache control section 240 can serve the data demand from the operand organizer C247 (drawing 2) and the operand organizer C248 (drawing 2) for coincidence through an operand bus interface. In

actuation, both operand both [one side or] 247 and 248 (drawing 2) offer an index 1874, and they take out the data demand signal 1876. [one of] The address-generation section 1881 generates one or the perfect external address 1877 beyond it to an index 1874. In order to investigate whether a related cache line is effective, the cache control section 1878 judges whether the demanded data exist in a cache 230 by inspecting the Rhine effective condition memory 1873, while inspecting the tag memory 1872 to the tag address of the generated address 1877. When the demanded data exist in cache memory 230, it is sent to the operation organizers 247 and 248 to whom the acknowledgement (response) signal 1879 relates with requested data 1880. When the demanded data do not exist in cache memory 230, the fetch of the demanded data 1870 is carried out from external memory through the input bus interface 1871 and the input interface switch 252 (drawing 2). The fetch of the data 1870 is carried out by offering the address 1877 with which the demand signal 1882 was outputted and the demanded data 1870 were generated. The acknowledgement signal 1883 and the demanded data 1870 are sent to the cache control section 1878 and cache memory 230, respectively. And the cache line relevant to the cache memory 230 is updated with the new data 1870. The tag address of a new cache line is also written in the tag memory 1872, and the Rhine effective condition 1873 in a new cache line is started. The acknowledgement signal 1879 is sent to the operand organizer 247 related with data 1870, or 248 (drawing 2).

[0463] The memory configuration of a data cache 230 is shown in drawing 142. A data cache 230 is arranged as a direct map cache in which cache line length has 128 cache lines C0, ..., C127 which are 32. the memory banks B0, ..., B7 in which separate addressing can do Cache RAM -- providing -- **** -- each memory bank -- that of 32 bits of 128 bank lines -- having -- each cache line Ci -- eight memory banks B0 and ... it has eight bank line B0i which corresponds in B7, ..., B7i.

[0464] The configuration of the generated external memory address is shown in drawing 143. The generated address is 32-bit WORD which consists of the 20-bit tag address, 7 bit-line addresses, the triplet bank address, and a 2-bit byte address. It is used for comparing the 20-bit tag address with the tag memorized by the tag address and the tag memory 1872. Seven bit-line addresses are used for the address of the related cache line in cache memory 1870. The triplet bank address is used for the address of MORIBANKU in which cache memory 1870 carries out ME relation. A 2-bit byte address is used for a cutting tool's address with which a 32-bit bank line is related.

[0465] Drawing 144 shows the block diagram of the structure of the data cache control section 240 and a data cache 230. Here, 128x256-bit RAM constitutes cache memory 230, and this consists of a memory bank in which eight 128x32-bit separation address attachments are possible. This RAM has the port (write) which can be written in, a write-in address port (write_addr), and a write-in dataport (write_data). Moreover, it has a reading possible port (read), eight reading address ports (read_addr), and eight reading data output ports (read_data). In order to enable the coincidence writing to all the memory banks of cache memory 230, the signal which can be written in is generated from the cache control block 1878. As occasion demands, a data cache 230 is updated by the data of Rhine beyond 1 or it from external memory through a write-in dataport (write_data). A write-in address port (write_addr) is provided with the Rhine address, and data of one line are written in by using the 8:1 multiplexing machine MUX. The 8:1 multiplexing machine MUX chooses the Rhine address from the external address generated under control of a data cache control section (addr_select). In order to make possible coincidence reading to all the memory banks of cache memory 230, a reading possible signal is generated from the cache control block 1878. By this approach, the data of eight different bank lines can be read into coincidence from eight reading dataports (read_data) according to each eight Rhine addresses of the memory bank of cache memory 230 with which write and an address port (read_addr) is provided.

[0466] The bank of each cache memory 230 has the programmable address-generation machine 1881. This makes possible concurrent access from eight related memory banks to eight different locations. Each address-generation machine 1881 has the dcc mode input for an operating mode setup of the address-generation machine 1881, an index packet input, a base address input, and address output. The operating mode of the programmable address-generation machine 1881 is the random access mode which the signal to (a) dcc mode input makes each address-generation machine 1881 the random access mode, and an external memory address is offered to an index packet input, and is outputted to the address output of one or the address-generation machine 1881 beyond it.;

(b) JPEG encoding to which the signal to a dcc mode input makes each address-generation machine 1881 the suitable mode, decode, a color space conversion, matrix multiplication mode. In this mode, the index to an index packet input is inputted into each address-generation machine 1881, and the index address is generated. The address-generation section can make an operating mode generate a maximum of eight different external

memory addresses.
[0467] The eight address-generation sections 1881 consist of the dcc modes and the indexes in which it has become from eight different logical circuits, and each has a base address as an input, and they have an external memory address as an output. A base address register 1885 memorizes the current base address which is the combination of an index packet, and the dcc mode register 1888 memorizes the current operating mode (dcc mode) of the data cache control section 240.
[0468] The tag memory 1872 consists of multiport (1 block and 128x20 bits) RAM. this RAM -- one -- writing -- a port (update-line-addr) and one -- it writes and has a possible port (write) and eight reading ports (tag0_data, ..., tag7_data). This makes the lookup of eight coincidence possible in a port (read0 line-addr, ..., read7 line-addr) by determining the tag address of Rhine of the memory address by which the current storage of the eight address-generation machines 1881 is carried out and which was generated more than one or it. The current tag address of these Rhine is outputted to the tag comparator 1886 from a port (tag0-data, ..., tag7-data). In order to enable the writing to the tag memory 1872 of a port (update-line-addr), a tag writing signal is generated by the need by the cache control block 1872.
[0469] The 128-bit Rhine valid memory 1873 is maintaining the valid condition of each cache line of cache memory 230. this -- one -- it writes and they are a port (update-line-addr) and one 128x1-bit memory which writes and consists of possible ports (update), eight reading ports (read0 line-addr, ..., read7 line-addr), and eight reading possible ports (linevalid0, ..., linevalid7). This makes possible eight coincidence lookups to a port (read0 line-addr, ..., read7 line-addr) like tag memory by making the Rhine valid condition saved to the eight address-generation sections 1881 in current Rhine to each Rhine address of the memory address generated more than one or it determine. The present Rhine valide bit of this Rhine is outputted to the tag comparator 1886 from a port (linevalid0, ..., linevalid7). The Rhine valid condition memory 1873 writes depending on the need, it writes for enabling the writing from the port (update-line-addr) to the Rhine valid condition memory 1873 to a port, and a signal generates from the cache control block 1878 in it.
[0470] The tag_data input for receiving the tag address by which current save is carried out in the tag memory 1872 of Rhine accessed by the Rhine address of the external address by which the tag comparator 1886 consists of eight tag comparators, and current generation was carried out, The tag_addr input of the tag address receptacle **** sake of the external memory address by which current generation was carried out, dcc_input for receiving the present mode-of-operation signal (dcc_mode) for setting up the tag address part compared, It has a line_valid input for receiving the Rhine valid condition by which current save is carried out in the Rhine valid condition memory 1873 in Rhine accessed by the Rhine address of the external address by which current generation was carried out. a comparator 1886 -- the eight address-generation sections 1881 -- it is alike, respectively, and it receives and has eight hit outputs. When the contents of the tag memory 1872 in the location accessed by the tag address of the generated external memory address and the Rhine address of the generated external memory are in agreement, a hit signal and the Rhine valid status bit 1873 to the Rhine are outputted. The DS saved to external memory in this example becomes small, and all the most significant bits of the tag address are the same. Therefore, what is necessary is to measure only the least significant bit from which the tag address changes. This becomes possible by setting up the present operating mode signal (dcc_mode) so that the tag comparator 1866 may measure the least significant bit from which the tag address changes.
[0471] When access to the data in cache memory 230 is possible for the cache control section 1878, it receives the demand (proc_req) and notice (proc_ack) from an operand B247 and an operand C248. The data of the address which is different from the bank to eight of cache memory 230 depending on a mode of operation are required. When requested data can access from cache memory 230, a hit is taken out from the tag comparator 1886 to Rhine of the memory. To the taken-out hit signal (hit0, ..., hit7), the cache control section 1878 generates the signal which can be read in a port (cache_read), and makes possible reading on the cache line from which the hit signal was taken out. When not a hit signal (hit0, ..., hit7) but the demand (proc_req) 1876 is advanced, the external memory address of the cache line of data is sent to the demand (ext_req) and ** which were generated at external memory. This cache line is written in eight banks of cache memory 230 through it, when it can input (ext_data). In this case, tag information is also written in the tag memory 1886 of the Rhine address, and the Rhine status bit 1873 of that Rhine is outputted.
[0472] The data from eight banks of cache memory 230 are outputted through some multiplexing machines which the data organizer 1892 has, and are positioned by the output data packet 1894 by the predetermined approach. The data organizer 1892 can choose and output 8-bit WORD by a certain mode of operation from eight 32-bit WORD outputted from eight memory banks by using the byte address (byte_addr) of the present

mode-of-operation signal (dcc_mode) and the generated external memory address. The data organizer 1892 does the direct output of the eight 32-bit WORD outputted from eight memory banks in other modes. A data organizer lines up and outputs to the method which was able to determine this data as mentioned above.
[0473] A demand is performed in the next phase.
1) A processing unit requires delivery packet data of the processing unit interface in the cache control section 1878 for the address.
2) Eight address-generation units 1881 generate the address of each block of cache memory according to a mode of operation.
[0474] 3) The tag location of the generated address is compared with the tag address saved to 4 blocks of the tag memory 1886 of three ports, and is positioned by the Rhine section equivalent to the eight generated addresses.
4) If they are in agreement and the Rhine valid condition 1873 of the Rhine is taken out, it will be considered that the demanded data exist in cache memory 230.
[0475] 5) The fetch of the data not existing is carried out through an external bus 1890, and eight blocks of cache memory 230 are updated by the contents of the data line from the external memory. The tag address of new data is written in the tag memory 1886, and the Rhine valid condition 1873 of the Rhine is taken out.
6) If all requested data exists in cache memory 230, it will appear in a processing unit in the decided packet format.
[0476] All the parts (drawing 2) of a co-processor 224 have included the Standard C Bus interface 303 (drawing 20) as mentioned above. The detail of the Standard C Bus interface register of the data cache control section 240 and a cache 230 is indicated from B42 by B46 of an Appendix B. A setup of this register controls actuation of the data control section 240. Since it is easy, only two registers (base_address and bcc_mode) are shown in drawing 153.
[0477] If the data cache control section 240 and a data cache 230 are effective, a data cache control section will make all cache lines an invalid at first, and will operate by the canonical mode. The data cache control section 240 and a cache 230 always change to the end of a certain instruction at a standard mode of operation. There is an option called "Auto-fill and validate" in all the modes except "Invalidate" mode. By setting 1 bit to dcc_cfg2 register, it can begin from the address to which all caches are saved by the base_address register. The data demand from the operand organizers B and C247,248 is stopped during this actuation. A cache becomes effective after this actuation finishes.
a. The external memory address of requested data is offered by two operand organizers in the mode of standard cache mode **. The address-generation section 1881 outputs an external memory address, and it confirms whether it exists in the memory cache 230 using internal tag memory. When both requested data does not exist in a cache 230, data are required from the input interface switch 252. It establishes in a continuous and instantaneous demand, and round robin scheduling is adopted.
[0478] It comes to be located in 32 bits behind the demanded data bus if one data item exists in a cache to an instantaneous demand. Other data are required outside through an input interface switch.
b. general single output color space conversion Mohd -- in this mode, a demand is advanced by a 12-bit cutting tool's address format from the operand organizer section B. A requested data item is a 8-bit color output value as shown in drawing 60. Twelve bit addresses are inputted into the index_packet input of the address-generation section 1881, and the eight address-generation sections 1881 generate the 32-bit external memory address of the format shown in drawing 96. A bank of this generated address, Rhine, and a byte address are determined by Table 12 and drawing 61. An external memory address is interpreted as the eight 9 bit lines and byte addresses, and it is used in order to point out the cutting tool of eight banks of RAM. A cache is accessed in order to calculate 8 byte values of the bank returned by the above-mentioned principle shown in the operand organizer section by the main data path 242 at drawing 60 for interpolation. Since they are settled in cache memory 230, before all general single output color value tables apply a single color translation mode, it is desirable to load a single output color value table to cache memory 230.
c. general multi-output color space conversion Mohd -- in this Mohd, a 12-bit word address can receive from the operand organizer section B247. A requested data item is the 32-bit color output value mentioned above with reference to drawing 62. Twelve bit addresses are inputted into the index_packet input of the address-generation section 1881, and the eight address-generation sections 1881 make eight different 32-bit external memory addresses of the format shown in drawing 96. Rhine of an external memory address and the tag address are determined by Table 12 and drawing 63. An external memory address is interpreted as eight bit addresses [nine] which have nine bit addresses divided into 7 bit-line addresses and the 2-bit tag address, as mentioned

above with reference to drawing 63. When the tag address is not discovered, a cache stops until suitable data are loaded from the input interface switch 252 (drawing 2). When data are available, output data are outputted to the operand organizer section.
d. JPEG coding Mohd -- in this Mohd, a table required for JPEG coding Mohd etc. is saved to a bank Of Cache RAM. Storage of a table is described at JPEG coding Mohd's (Tables 14 and 16) place.
e. low-speed JPEG decode Mohd -- in this Mohd, data are generated according to Table 17.
f. matrix multiplication Mohd -- in this Mohd, a cache is used in order to access the data of 256-byte Rhine.
g. Disabled Mohd -- all demands are passed by the input interface switch 252 in this Mohd.
h. Invalidate (nullification) Mohd -- in this Mohd, the contents of all the caches are made into an invalid by clearing the Rhine valid status bit.
[0479] 3.18.7 By input interface switch drawing 2, an input interface switch achieves **** which adjusts the requested data from the pixel organizer section 246, the data cache control section 240, and the instruction control section 235. Moreover, this transmits the address and data required for the external interface control section 238 and the local memory control section 236.
[0480] The input interface switch 252 saves the setup at one register of the memory objects in a base address or a host memory map. Since 20 address bits are required, this is the bar CHUARU address which aligns in a page boundary. The input interface switch 252 subtracts the base address bit of a co-processor from 6 bits of high orders of the starting address of data first to the demand from the pixel organizer section, a data cache control section, and an instruction control section. It means that this result is negative, or a PCI bus is a desirable transmission place when 6 bits of high orders of this result are not 0.
[0481] When 6 bits of high orders of a result are 0, it means that a data map expresses the memory location of a co-processor. Then, an input interface switch inspects the following triplet, in order that the location of a co-processor may distinguish whether it is the right. The just location of a co-processor is 16 megabytes which the common interface which begins from offset 0x01000000 occupies from the base address of one co-processor.
[0482] 2) 32 megabytes which the local memory control section (LMC) which begins from offset 0x02000000 occupies from the base address of the memory object of a co-processor. It is considered by the input interface switch that the demand which points out the location of an unjust co-processor is an error. A PCI bus serves as a data source of the addresses other than the field which the memory object of a co-processor occupies. An input interface switch uses i source signal in order to tell EIC about the thing from it and a common interface for whether requested data is a thing from a PCI bus.
[0483] A just demand is transmitted to a suitable IBus interface after address decode processing. EIC and LMC transmit data to an input interface switch, when an i-ack signal is taken out. however, since an input interface switch does not count the numbers of words inputted, if the i-oe signal with which current data transmission is controlled [when it finishes and] by the pixel organizer section, an instruction control section, and a data cache control section do not have ******, it will not become.
[0484] The input interface switch 252 adjusts three modules, the pixel organizer section, a data cache control section, and an instruction control section. Although these can require data of coincidence, since there are only two physical resources, the demand is not processed soon. The accommodation technique used for an input interface switch uses priority as the base, and a program is also possible for it. The control bit in the setting register of an input interface switch specifies the relative priority of an instruction control section, a data cache control section, and the pixel organizer section. The demand from a module with low priority is accepted when there is no access request to the same resource from other two modules. If the same priority as at least two demand issue origin is given, in order to determine the issue origin by which a demand is received, it will be necessary to use a round robin technique.
[0485] Since it is impossible to access the one source immediately, it is necessary to see an input interface switch whether prefetch the data which memorized the address and burst length of data who were demanded and were offered from the requiring agency. In the processing to a certain source, when there is no IBus processing, the adjustment processing which determines priority is needed.
[0486] The detail of the instruction interface switch 252 is shown in drawing 145. A switch 252 has two IBus transceivers 661 between the address decoder 863 and a controller 864 in addition to a Standard C Bus interface and register file 860. The address decoder 863 carries out the address decode to the demand received from the pixel organizer section, the data cache control section, and the instruction control section. The address decoder 863 inspects whether the address is just, and also carries out remapping of the address as occasion demands. A controller 864 decides which demand to transmit to the IBus transceiver 662 from the IBus transceiver 661.

Priority is programmable.
[0487] The IBus transceivers 861 and 862 have multiplexing, the demulti pre KUSHINGU function, and the buffer ring function of the tri-state for enabling the communication link to the input interface switch from other interfaces.
3.18.8 Set to local memory control-section drawing 2, and the local memory control section 236 takes charge of all the processings of control of a local memory, and the access request between a local memory and the module in a co-processor. The local memory control section 236 answers the write request from the result organizer 249, and a read-out demand from the input interface switch 252. Furthermore, it answers also to the peripheral-interface-adapter control section 237, read-out from the usual general CBus input, and a write request. The local memory control section uses the programmable priority system, and in order to maximize a throughput further, it has adopted the FIFO buffer.
[0488] In this invention, in order to carry out the decouple of the port other than a first in first out (FIFO) buffer from a memory array, the multiport burst dynamic memory control section is used. Drawing 146 shows the block diagram of 4 port burst dynamic memory control section according to the 1st example of this invention. Two write-in ports (A1944 and B1946) and two read-out ports (C1948 and D1950) which need access to a memory array 1910 for this circuit are included. To coming out from a memory array 1910 by separate FIFO1936 and 1938 courses, the data path from two write-in ports passes along separate FIFO 1920 and 1922, and the data path of the read-out ports 1948 and 1950 goes to a memory array 1910 by multiplexing section 1912 course. The CC section 1932 adjusts the whole port access while driving all control signals required for the interface to dynamic memory 1910. The refresh counter 1934 determines the need stage of the refresh cycle of dynamic memory for a memory array 1910, and adjusts these with a control section 1932.
[0489] read-out and the writing of preferably as opposed to a memory array 1910 of data -- FIFO 1920 and 1922 from the write-in ports 1944 and 1946 -- or it reads from FIFO 1936 and 1938 and is carried out by transfer twice the rate of ports 1948 and 1950. Consequently, time amount (it is the bottleneck of what kind of memory system) which the transfer from a memory array 1910 or the transfer to a memory array 1910 takes is shortened as much as possible to the time amount taken to read with writing and to transmit data through ports 1944, 1946, 1948, and 1950.
[0490] Data are written in a memory array 1910 via either of the write-in ports 1944 and 1946. The circuit connected to the write-in ports 1944 and 1946 will recognize only FIFO 1920 and 1922 of initial value zero. The data transfer which lets the write-in ports 1944 and 1946 pass progresses smoothly until FIFO 1920 and 1922 fills or a burst is completed. If data are first written in FIFO 1920 and 1922, a control section 1932 will perform mediation with other ports for access to DRAM. If access is obtained, data will be read from FIFO 1920 and 1922 at the highest rate, and will be written in a memory array 1910. The burst write cycle to DRAM1910 is started, when a number of data word by which presetting was carried out to FIFO 1920 and 1922 is saved, or only when the burst from a write-in port is completed. It continues until it progresses from the time of a permission being granted in the case of which, and FIFO 1920 and 1922 becomes empty to it or the burst to DRAM1910 has the cycle demand from a higher priority port in it. From a write-in port, it is not interfered to FIFO 1920 and 1922 and is continuously written in until FIFO is full of data, or a current burst is completed and a new burst begins also in which event. In the case of the latter, a new burst does not advance until a former burst empties FIFO 1920 and 1922 and is written in DRAM1910. In the case of the former, data transfer is resumed, as soon as the first WORD is read from FIFO 1920 and 1922 and is written in DRAM1910. Since the data transfer from FIFO 1920 and 1922 is the highest rate, the write-in ports 1944 and 1946 are able to carry out a urinal stall, only when a control section 1832 is interrupted by the cycle demand from other ports. It is desirable to maintain any interruption to the data transfer from the write-in ports 1944 and 1946 to FIFO 1920 and 1922 to min as much as possible.
[0491] The read-out ports 1948 and 1950 operate in reverse order. If the read-out ports 1948 and 1950 read and a demand is advanced, a DRAM cycle will be required at once. If the authorization to this demand is obtained, a memory array 1910 will be read and data will be written in corresponding FIFO 1936 and 1938. As soon as the first data word is written in FIFO 1936 and 1938, read-out by the read-out ports 1948 and 1950 becomes possible. Thus, although initial delay exists in obtaining the first data word, probably the delay beyond it does not come out for acquisition of the data word which after that follows. Read-out of DRAM will be ended, when read-out FIFO 1936 and 1938 becomes [whether there is any DRAM demand of a higher priority, and], or if the read-out ports 1948 and 1950 stop requiring data more. After once doing in this way and completing read-out, it is not resumed until a leeway is given in the number of the data word by which presetting is carried out to

FIFO 1936 and 1938. Once a read-out port ends a cycle, any data which remain in FIFO 1936 and 1938 will be discarded.

[0492] in order that DRAM control may always exceed the minimum value, re-mediation to DRAM access is restricted so that a burst may interrupt and may not be carried out, until a number of all data word by which presetting is carried out is transmitted, and read-out FIFO 1936 and 1938 becomes [whether corresponding FIFO 1920 and 1922 becomes empty and] or. All the access ports 1944, 1946, 1948, and 1950 have a burst starting address corresponding to each, and these are latched to the counter 1942 at the time of initiation of a burst. This counter can be resumed by the right memory address also by ****, even if it holds the current address for the dealings to a port and a metaphor transfer is interrupted. Only the address of a DRAM cycle [AKUTIVU now] is chosen by the multiplexing section 1940, and is sent to the line address counter 1916 and the train address counter 1918. the address -- low -- N [degree] bit is inputted into the train counter 1918, and one upper address bit is inputted into a line-counter 1916. The multiplexing section 1914 outputs a line address to a memory array 1910 from a line-counter 1916 between the line address times of DRAM, and sends the train address from the train counter 1918 between the train address times of DRAM. The line address counter 1916 and the train address counter 1918 are loaded to a memory array DRAM 1910 at the time of initiation of what kind of burst. This is a fact which is [at both times of initiation of a port cycle, and continuation of the interrupted burst] applied. After the transfer to each memory breaks out, the increment of the train address counter 1918 is carried out, and if the train address counter 1918 changes to zero, the increment of the line address counter 1916 will be carried out. In the case of the latter, a burst must be ended, and it must be resumed by the new line address.

[0493] In this example, the memory array 1910 includes 4x octet-bit-byte Rhine, and assumes that 32 bits per WORD are constituted. Furthermore, there are sets 1950 and 1952 of 4 bytes of write-in enable signal corresponding to each write-in port 1944 and 1946, and data are individually written in each 8-bit part of each 32-bit data word in a memory array 1910. Since it is possible to cover the mask to the writing of data over what kind of cutting tool in each WORD written in a memory array 1910 at arbitration, it is necessary to write in corresponding FIFO 1926 and 1928 with each data word, and to store enabling information. Although such FIFO 1926 and 1928 is controlled by the same signal as being used for control of write-in FIFO 1920 and 1922, only 4 bits is used instead of 32 bits needed for the writing of the data to FIFO 1920 and 1922. Similarly, the multiplexing section 1930 is controlled like the multiplexing section 1912. The writing to the WORD to which the address of [in a memory array 1910] was carried out synchronizing with the data write-in [selected] in which a control section is inputted into a memory array 1910 by the multiplexing section 1912 by writing in and inputting enabling into a control section 1932 using such information is alternatively made possible or impossible.

[0494] The configuration of drawing 146 operates under control of a control section 1932. Drawing 147 is a state diagram showing the detail of actuation of a control section 1932 in drawing 146. the power-up back and the time of completion of reset -- a condition machine -- compulsory -- IDLE100 condition -- becoming -- this condition -- all DRAM control signals -- being inactive (high) -- becoming, the multiplexing section 1914 sends a line address to the DRAM array 1910. If refresh or a cycle demand is detected, it will change to RASDEL11962 condition. If a cycle demand and refresh are lost by the following clock edge, a condition machine will return to IDLE1900 condition. Otherwise, DRAM When a tRP (RAS precharge timing limit) period is filled, it changes to RASON1966 condition, and the row-address strobe signal RAS is set to a low level at this time. After tRCD (delay timing limit to CAS from RAS) is filled, it changes to COL1968 condition, and the multiplexing section 1914 is switched so that the train address for inputting into the DRAM array 1910 may be chosen. CASON1970 condition changes by the following clock edge, and a DRAM column address strobe (CAS) signal becomes an active low. If tCAS (CAS AKUTIVU timing limit) is filled, it will change to CASOFF1972 condition and a DRAM column address strobe (CAS) will once become in bitter taste tee VUHAI again in this condition. while the further data word is to be transmitted, or neither the cycle demand of a higher priority nor refresh is approaching here -- or -- re--- when too quick for arbitrating, once a tCP (CAS precharge timing limit) period is filled, it will return to CASON1970 condition, and a DRAM column address strobe (CAS) becomes bitter taste tee VURO again. If both tRAS (RAS AKUTIVU timing limit) and tCP (CAS precharge timing limit) are filled when re-mediation occurs and the cycle demand of a higher priority and refresh are approaching, or there is no transfer of the further data word instead, it will change to RASOFF1974 condition. In this condition, a DRAM row-address strobe (RAS) signal becomes in bitter taste tee VUHAI. A condition machine returns to IDLE1860 condition by the following clock edge, and the next cycle initiation is

prepared.
[0495] RASDEL2 It is RCASON once tRP (RAS precharge timing limit) will be filled, if a refresh demand is detected in the 1964 condition. 1980 conditions change. A DRAM column address strobe becomes bitter taste tee VURO in this condition, and it is DRAM before [a refresh cycle of] RAS. CAS is started. Transition is RRASON at the following clock edge. It is carried out 1978 and a DRAM row-address strobe (RAS) becomes bitter taste tee VURO. Transition is RCASOFF when tCAS (CAS AKUTIVU timing limit) is filled. It is carried out 1976 and a DRAM column address strobe (CAS) becomes in bitter taste tee VUHAI. Once tRAS (RAS AKUTIVU timing limit) is filled, transition will be performed RASOFF1974, and a DRAM row-address strobe (RAS) becomes in bitter taste tee VUHAI, and makes an effective target end a refresh cycle. A condition machine continues the above behavior for the usual DRAM cycle, and is IDLE. It changes to 1960 conditions.
[0496] The refresh counter 1934 of drawing 146 is a counter simply, and generates a refresh demand signal per 15 microseconds at 1 time of a fixed rate, or the rate which became settled by demand of a special DRAM contractor. If a refresh demand is published, this demand will continue an issue condition until it is recognized with the condition vessel of drawing 147. This acknowledgement is performed when a condition machine goes into RCASON1980 condition, and it continues that condition until a condition machine detects withdrawal of a refresh demand.
[0497] Actuation of the mediation machine 1924 of drawing 146 is shown to drawing 148 by pseudocode form. Here, in order to maintain the fairness to access, the mechanism which corrects the priority of a cycle claimant is described to be the approach of deciding which access to a memory array 1910 is permitted in four cycle demand publishers. The symbol used for these codes is explained in drawing 149.
[0498] Each demand publisher has 4 bits showing the priority of the demand. Presetting of the 2 bits is carried out to the general priority by the configuration value set as the configuration register with a common high order. 2 bits of low order of a priority are stored in two bit counters updated by the arbitrator 24. An arbitrator 1924 permits access to the claimant of a peak price [the time of deciding a mediatory winner / only] for the value of 4 bits of each claimant. If a cycle is permitted to a claimant, the value of the priority counter of 2 bits of low order will become zero, and the increment of all the priority counts of 2 bits of low order of other claimants with the priority value of the 2 bits of the same high orders and the priority value of 2 bits of low order lower than a winner will be carried out every [1].

Since it became timeout time, translation result display processing is stopped.

state diagram showing the detail of actuation of a control section 1932 in drawing 146. the power-up back and the time of completion of reset -- a condition machine -- compulsory -- IDLE100 condition -- becoming -- this condition -- all DRAM control signals -- being inactive (high) -- becoming, the multiplexing section 1914 sends a line address to the DRAM array 1910. If refresh or a cycle demand is detected, it will change to RASDEL11962 condition. If a cycle demand and refresh are lost by the following clock edge, a condition machine will return to IDLE1900 condition. Otherwise, DRAM When a tRP (RAS precharge timing limit) period is filled, it changes to RASON1966 condition, and the row-address strobe signal RAS is set to a low level at this time. After tRCD (delay timing limit to CAS from RAS) is filled, it changes to COL1968 condition, and the multiplexing section 1914 is switched so that the train address for inputting into the DRAM array 1910 may be chosen. CASON1970 condition changes by the following clock edge, and a DRAM column address strobe (CAS) signal becomes an active low. If tCAS (CAS AKUTIVU timing limit) is filled, it will change to CASOFF1972 condition and a DRAM column address strobe (CAS) will once become in bitter taste tee VUHAI again in this condition. while the further data word is to be transmitted, or neither the cycle demand of a higher priority nor refresh is approaching here -- or -- re--- when too quick for arbitrating, once a tCP (CAS precharge timing limit) period is filled, it will return to CASON1970 condition, and a DRAM column address strobe (CAS) becomes bitter taste tee VURO again. If both tRAS (RAS AKUTIVU timing limit) and tCP (CAS precharge timing limit) are filled when re-arbitration occurs and the cycle demand of a higher priority and refresh are approaching, or there is no transfer of the further data word instead, it will change to RASOFF1974 condition. In this condition, a DRAM row-address strobe (RAS) signal becomes in bitter taste tee VUHAI. A condition machine returns to IDLE1860 condition by the following clock edge, and the next cycle initiation is prepared.

[0495] RASDEL2 It is RCASON once tRP (RAS precharge timing limit) will be filled, if a refresh demand is detected in the 1964 condition. 1980 conditions change. A DRAM column address strobe becomes bitter taste tee VURO in this condition, and it is DRAM before [a refresh cycle of] RAS. CAS is started. Transition is RRASON at the following clock edge. It is carried out 1978 and a DRAM row-address strobe (RAS) becomes bitter taste tee VURO. Transition is RCASOFF when tCAS (CAS AKUTIVU timing limit) is filled. It is carried out 1976 and a DRAM column address strobe (CAS) becomes in bitter taste tee VUHAI. Once tRAS (RAS AKUTIVU timing limit) is filled, transition will be performed RASOFF1974, and a DRAM row-address strobe (RAS) becomes in bitter taste tee VUHAI, and makes an effective target end a refresh cycle. A condition machine continues the above behavior for the usual DRAM cycle, and is IDLE. It changes to 1960 conditions.

[0496] The refresh counter 1934 of drawing 146 is a counter simply, and generates a refresh demand signal per 15 microseconds at 1 time of a fixed rate, or the rate which became settled by demand of a special DRAM contractor. If a refresh demand is published, this demand will continue an issuance condition until it is recognized with the condition vessel of drawing 147. This acknowledgement is performed when a condition machine goes into RCASON1980 condition, and it continues that condition until a condition machine detects dislodging of a refresh demand.

[0497] Actuation of the arbitration machine 1924 of drawing 146 is shown to drawing 148 by pseudocode form. Here, in order to maintain the fairness to access, the mechanism which corrects the priority of a cycle claimant is described to be the approach of deciding which access to a memory array 1910 is permitted in four cycle demand publishers. The symbol used for these codes is explained in drawing 149.

[0498] Each demand publisher has 4 bits showing the priority of the demand. Presetting of the 2 bits is carried out to the general priority by the configuration value set as the configuration register with a common high order. 2 bits of low order of a priority are stored in two bit counters updated by the arbitrator 24. An arbitrator 1924 permits access to the claimant of a peak price [the time of deciding a mediatory winner / only] for the value of 4 bits of each claimant. If a cycle is permitted to a claimant, the value of the priority counter of 2 bits of low order will become zero, and the increment of all the priority counts of 2 bits of low order of other claimants with the priority value of the 2 bits of the same high orders and the priority value of 2 bits of low order lower than a winner will be carried out every [1]. Consequently, the claimant to which access to a memory array 1910 was permitted now becomes the lowest priority between claimants with the same 2 bit priority value of high orders. The priority value of 2 bits of low order of a claimant in which the priority value of 2 bits of high orders has a value different from a winner is not influenced. The value of 2 bits of high orders of a priority determined the general priority of a claimant, and the value of 2 bits of low order has realized the fair arbitration scheme between the claimants of the same high order priority. the exchange partial from a fixed priority (2 bits of high orders of each claimant are unique) by which connection was carried out by hardware by using this

scheme, and partial hardware connection (although it is not all, some 2 bit priorities of high orders differ from those [other]) -- fair various arbitration schemes to changing (the priority values of 2 bits of all high orders being the same) are strictly realizable.
[0499] Drawing 149 shows the structure over each claimant and the directions of a bit of a priority bit. Here, the semantics of the symbol used for drawing 148 is also defined. Various kinds of FIFO 1920, 1922, and 1938 in the above-mentioned example and 1936 are 32 words in width of face of 32 bits, and depth. This depth has given compromise by the good line between effectiveness and the circuit area consumed. However, the value of the depth is changed according to the needs of specific application with change of performance.
[0500] Moreover, 4 configurations of port shown here are only one example. effectiveness is acquired carrying out reading appearance to a memory array, or preparing a single FIFO buffer between either of the write-in ports. However, when it writes in with read-out of a large number and a port is used, the highest improvement in speed will be obtained.
3.18.9 The module 239 besides other modules performs generating of the clock for interfacing between actuation of a co-processor 224, a reset synchronization, the error by turning an internal diagnostic signal to an external pin if needed, multiplexing of an interruput signal and the interior of CBus, and external form, multiplexing to general / external Cbus output pin of the interior and a general Bus signal, etc., and selection. Of course, actuation of the other modules 239 changes with the demands to crocking and embodiment details by the ASIC technology used.
[0501] 3.18.10 The external interface control section, next the description of this invention described relate to the approach and equipment for offering virtual memory with the host computer which has the co-processor which shares virtual memory. It is groping for the example of this invention so that a co-processor may be interlocked with a host processor and actuation of it may be attained in virtual memory mode.
[0502] Especially a co-processor can be operated in the virtual memory mode of a host processor. The virtual memory pair physics memory-mapping device which can refer to the virtual memory table of a host processor is contained in the co-processor, and the instruction address generated by the co-processor is mapped in the physical address with which it corresponds in the memory of a host processor. Rather, a virtual memory pair physics memory-mapping device forms a part of computer graphic co-processor, in order to generate a graphical image. The module of a large number which can perform various complicated actuation is contained in an image at a co-processor. A mapping device participates in the interaction between a co-processor and a host processor.
[0503] The external-interface control section (EIC) 238 is PCI of a co-processor. Bus and the interface through which it passes general Bus are offered. Furthermore, an external-interface control section also offers the memory management which connects between the internal virtual address space of a co-processor, and the physical address spaces of a host system. The external-interface control section 238 is PCI when writing data in host memory according to the demand from an organizer 249 a result, the time of reading data from host memory according to the demand from the input interface switch 252, and. It operates as a master on Bus. PCI Access to Bus is "PCI Local Bus Specification, draft2.1" PCI. special interest It embodies according to group and the criterion of 1994.
[0504] The external-interface control section 238 arbitrates the simultaneous demand for the PCI dealings from an organizer 249 as a result of the input interface switch 252. As for arbitration, to be able to constitute is desirable. Cache line read-out [a host co-processor] of one or less line, one line of a host and read-out of the cache line between two lines, and read-out of two lines or the cache line beyond it are included in the received type of a demand at once. Unrestricted writing is also embodied by the external-interface control section 238 in length. Furthermore, the external-interface control section 238 also performs PURIFETCHINGU of data optionally.
[0505] The memory management which offers the address mapping from virtual memory to a host's physical memory for the intramodule of all co-processors is included in construction of the external-interface control section 238. This mapping is completely transparent to the module which requires access. If the external-interface control section 238 receives the access request to host memory, a memory management unit will be initialized and the demanded address will be changed. When a memory management unit fails in conversion of the address, depending on the case, it is one or PCI beyond it. A result to which dealings of Bus(es) complete conversion of the address is brought. For this, the memory management unit itself is PCI. It means getting used to another source which requires dealings of Bus. If the burst demanded by the organizer 249 as a result of the input interface switch 252 crosses the boundary of a virtual page, the external-interface control section 238 will

operate a memory management unit automatically, and will redo mapping of all the virtual addresses correctly.
[0506] As for the memory management unit (MMU) (915 of drawing 150), 16 look-aside buffers (TLB) have been bases. TLB operates as a cache of virtual pair physics address mapping. In TLB, the following activities are possible.
1) Comparison : if the virtual address is given, TLB will return a corresponding physical address or a TLB mistake signal (when there is no effective entry which matches the address).
[0507] 2) Permutation : new virtual pair physical mapping is written in TLB instead of the existing entry or the entry which is not effective.
3) -- nullification: -- when the virtual address is given, the entry which matched when the entry of TLB was matched is cancelled.
4) All nullification : cancel all TLB entries.
[0508] 5) Read-out : reading appearance of imagination and the physical address of a TLB entry is carried out with 4 bit-address base. It is used only for a test.
6) Writing : imagination and the physical address of a TLB entry are written in with 4 bit-address base.
The entry in TLB is a format as shown in drawing 151. Each effective entry consists of a physical address 671 of 670 or 20 bits of 20-bit virtual addresses, and a flag showing the ability of a corresponding physical page to be written in. The allowance page size of an entry is 4 K bytes. The register in MMU can be used for covering a mask over the address to 10 bits used for the comparison. The page of TLB is supported by this to 4 M bytes. Since the number of mask registers is one, refer to the page of the same size for all TLB entries.
[0509] The "Least-Recently Used" (LRU) permutation algorithm is used for TLB. A new entry is overwritten by the entry in which the longest time amount passed. It is because it was written in at the end or was in agreement by the comparison activity. This is applied only when there is no invalid entry. When there is an invalid entry, before overwriting an effective entry, it is written in an invalid entry.
[0510] Drawing 152 shows the flow of TLB comparison actuation. The received virtual address 880 is divided into three parts of 881-883. 12 bits of low order, since 881 is always the part of the offset in a page, it is direct sent to the corresponding physical address bit 885. 10 bits, 882 is the part of offset by the page size, or the part of the page number as [following] it was set up by the mask bit. Since a bit is the part of a page offset, the value of the zero in a mask register 887 shows that it must not use for a TLB comparison. "ANDED" (AND) of the 10 address bit is logically carried out to ten mask bits, and it gives the virtual page number 889 of 10 bits of low order for a TLB lookup. 883 is direct used as 10 bits of high orders of a virtual page number 889 10 bits of high orders of the virtual address.
[0511] Thus, the generated 20-bit virtual page number is sent to TLB. If this is in agreement with one of the entries, TLB will return the number of the location which was in agreement with the corresponding physical-page number 872. A physical address 873 is generated from a physical-page number, using a mask register 887 again. 10 bits of high orders of the physical-page number 872 are direct used as 10 bits of high orders of a physical address 873. 10 bits next to a physical address 872 -- a physical-page number (when a corresponding mask bit is 1) or the virtual address (when a mask bit is 0) -- it is chosen from that either as 875. 885 is direct given from the virtual address 12 bits of low order of a physical address.
[0512] Finally, according to a match, the LRU buffer 876 is updated and expresses the activity of the address with which it was matched. A TLB mistake is generated when an organizer 249 demands access to the virtual address which does not exist in TLB872 as a result of the input interface switch 252. In this case, before MMU advances processing of demanded access, it must fetch the virtual pair physics conversion demanded from the page table of the host memory 203, and must write it in TLB.
[0513] A page table is a hash table of host main memory. Each page table entry consists of two 32-bit WORD of a format as shown in drawing 153. The 2nd WORD constitutes 20 bits of high orders for a physical address, and 12 bits of low order are reserved. 20 bits of high orders of the corresponding virtual address are given to the first WORD. The validity (V) bit, and (W) or the "lead-only" bit which can be written in is contained in 12 bits of low order, and the remaining 10 bits are reserved.
[0514] The same information as a TLB entry is fundamentally included in the page table entry. The flag of the excess of a page table is reserved. The page table itself is usually distributed over two or more pages in main memory 203, generally the virtual space is adjoined, and physical space has not touched. The set of the page table pointer of 16 set up by software is included in MMU, and each is a 20-bit pointer to the 4-K byte memory area containing the part of a page table. It means that a co-processor 224 supports the page table of 64K byte size, and this has 8 K page mapping. In the system of a 4-K byte page size, this means a maximum of 32 M

bytes of mapped virtual address space. A page table pointer is always referring to 4 K bytes of memory area regardless of the page size used for TLB rather.
[0515] The MMU actuation after a TLB mistake is shown in 690 of drawing 154 as follows.
1. Perform the hash function 892 on the virtual page number 891 which does not exist in TLB, and generate a 13-bit index to a page table.
2. Choose the page table pointer 895 using 894 4 bits of high orders of the page table indexes 894 and 896.
[0516] 9 bits of low order of the 3.20-bit page table pointer 895 and the page table index 896 are connected, and by setting 000 as the lowest triplet, (in order that a page table entry may occupy 8 bytes of host memory), the physical address 890 of the demanded page table entry is generated.
4. Begin from the physical address 898 of a page table entry, and read 8 bytes from host memory.
[0517] When 5.8 bytes of page table entry 900 is returned to a PCI bus, if the VALID bit is set to 1, a virtual page number will be compared with the virtual page number of the origin which caused the TLB mistake. If both do not match, the fetch of the following page table entry will be carried out using the above-mentioned process (the increment of every 8 bytes of physical address is carried out). This process is continued until the page table entry of the virtual page number which matches is found, or until it encounters an invalid page table entry. When an invalid page table entry is encountered, a page fault error is issued and processing is stopped.
[0518] 6. If the page table entry which has the virtual page number which matches is found, a perfect entry will be written in TLB by permutation actuation. A new entry is put on the TLB location pointed at by the LRU buffer 876. And the comparison activity of TLB is done again and processing of host memory access in which origin was required favorably is attained. If a new entry is written in TLB, the LRU buffer 876 will be updated.
[0519] The hash function 892 embodied by EIC238 uses the following equation to a 20-bit virtual page number (vpn).
index=((vpn>>S1)XOR(vpn>>S2)XOR(vpn>>S3))&Ox1fff;
Here, S1, S2, and S3 are shift amounts (forward or negative) programmable in independent, and they can take four values, respectively.
[0520] If the linear search of a page table crosses the boundary which is 4 K bytes, MMU will choose the following page table pointer automatically, and will continue retrieval in a right physical memory location. Wrapping from the last of a page table to the beginning is included in this activity. **-JITEBURU always contains at least one invalid (null) entry so that retrieval may always be ended.
[0521] The entry corresponding to the page which added the page table entry for a virtual page new whenever software permutes the page in host memory, and was permuted must be deleted. Moreover, the cache of the old page table entry must not be carried out to TLB of a co-processor 224. This is performed by achieving the TLB nullification cycle in MMU.
[0522] A nullification cycle carries out the virtual page number cancelled with the bit which causes nullification, and is achieved through the register writing to MMU. This register writing is achieved through the instruction interrupted by direct or the instruction decoder by software. Nullification is achieved on TLB for the offered virtual page number. If a TLB entry is matched, a LRU table will be updated so that an entry may be marked on an invalid and the cancelled location may be used in the next permutation.
[0523] Undecided nullification has a priority higher than what kind of undecided TLB comparison. If nullification is completed, MMU will clear a nullification bit and it will tell that the next nullification processing is possible. This is called page fault when the effective page table entry for the virtual address with which MMU was demanded cannot be found. MMU takes out an error signal and keeps the virtual address from which the fault was started to a register with accessible software. MMU goes into an idle state, and it stands by until an error is solved. **** [a clearance of interruption / begin / again / MMU / from the dealings as which the degree was required / an activity]
[0524] A page fault is taken out also when the write-in activity to the page (it is not marked that writing is possible) marked as it is read-only is made. The dealings demand from an organizer 249 responds to the external-interface control section (EIC) 238 as a result of the input interface switch 252 by which the address is carried out to the common bus. a demand current in each demand module -- the object for common buses -- or the object for PCI buses is expressed. Unlike using a common bus, the EIC actuation to a common bus demand is divided into communication with an organizer 249 as thoroughly as the actuation to a PCI demand as a result of the input interface switch 252. Furthermore, the Cbus dealings type which carries out the address to common bus space direct also responds to EIC238.
[0525] Drawing 150 shows the structure of the external-interface control section 238. An IBus demand passes

along the multiplexing section 910, and the multiplexing section 910 leads a demand to suitable (PCI or common bus) intramodule based on the destination of a demand. A demand into a common bus is sent to the general bus control section 911 also with RBus and CBus. In order that the common bus on RBus and a PCI bus demand may use a different control signal, the multiplexing section is not needed for this bus.

[0526] The IBus demand led to the PCI bus is treated by the IBus driver (IBD) 912. Similarly, the RBus demand to PCI is processed by the RBus receiver (RBR) 914. IBD912 and RBR914 send the virtual address to the memory management unit (MMU) 915 which returns a physical address. IBD, RBR, and MMU can require a PCI transaction, respectively, and these are generated and controlled by the PCI master mode control section (PMC) 917. IBD and MMU require only PCI read-out and RBR requires only PCI writing.

[0527] The separate PCI target modal-control section (PTC) 918 processes all the PCI transactions by which the address was carried out as a target to the co-processor. This enables access to delivery and all other modules for a CBus master mode signal to an instruction control section. In order that PTC may send the returned CBus data to a PCI bus via PMC, control of a PCI data bus pin is taken out from the single source.

[0528] The CBus transaction by which the address was carried out to the EIC register to module memory is treated with the Standard C Bus interface 7. All submodules get a bit from a control register, and return a bit to a status register. These are located in the interior of a Standard C Bus interface. The parity generation and the check for a PCI bus transaction are processed by the parity generation and the check (PGC) module 921 which operate under control of PMC and PTC. The generated parity is sent to a PCI bus like a parity error signal. The result of a parity check is sent also to the configuration register of PTC for an error report.

[0529] Drawing 155 shows the structure of the IBus driver 912 of drawing 150. The accepted IBus address and a control signal are 930 latched at the starting point of a cycle. OR gate 931 detects the beginning of a cycle and generates a start signal in the control logic 932. The upper address bit of the latch 930 who forms a virtual page number is loaded to a counter 935. A virtual page number is sent to MMU915 (drawing 150) which returns the physical-page number latched to 936.

[0530] A physical-page number and a low order virtual-address bit are recombined with a mask 937, and form the address 938 for a PCI demand to PMC717 (drawing 102). Moreover, the burst count for a cycle is also loaded to a counter 939. Prefetch actuation uses a comparison circuit 943 with a different counter 941 and a different address latch. The data returned from PMC are loaded to FIFO944 with the marker to whom data express whether it is a part of pre-planned system. If data become usable in the part in front of FIFO944, it will read by latch 945 and 946 courses, and clock out will be carried out by logic. The read-out logic 946 also generates an IBus acknowledgement signal.

[0531] The central control block 932 controls sequential processing of all addresses and data elements and the interfaces to PMC including a condition machine. The virtual page number counter 935 is a page number bit from the IBus address, and is loaded with initiation of an IBus transaction. 10 bits of high orders of these 20 bit counters come from the always accepted address. To 10 bits of low order, each bit is loaded from the address which will be accepted if the corresponding mask bit 937 is set to 1, otherwise, a counter bit is set to 1. The value of 20 bits is sent to an MMU interface.

[0532] In the usual actuation, a virtual page number is not used after initial address translation. However, when IBD detects page boundary **** of a burst, the increment of the virtual page counter is carried out, and another conversion is performed. Since the lower bit which is not a part of virtual page number is set to 1 when a counter is loaded, a simple increment with a value of 20 bits brings about the increment of the actual page number field. After an increment is carried out, in order to set up a counter for the next increment, a mask bit 937 is used again.

[0533] A physical address is 936 latched whenever MMU returns an effective physical-page number after conversion. A mask bit is used in order to combine correctly the returned physical-page number and the original virtual-address bit. The physical address counter 938 is loaded by the physical address latch 936. Whenever WORD is returned from PMC, the increment of this is carried out. Whenever an increment is carried out, the monitor of the counter is carried out, and it judges whether the transaction is going to cross the page boundary. A mask bit is used for judging which bit of a counter is used for a comparison. If it detects that the number of WORD with which the counter remains in the page is two or less, a signal will be taken out to the control logic 932, a PCI demand current [after / of two / data transfer] will be ended, and new address translation will be required if needed. A counter is again loaded after new address translation, and a PCI demand resumes it.

[0534] The burst counter 939 is a 6-bit down counter loaded with an IBus burst value at the starting point of a transaction. Whenever WORD is returned from PMC, the decrement of this is carried out. If the value of a

counter is set or less to two, a signal is taken out to the control logic 932, and a PCI transaction can be ended now after two data transfer (unless PURIFETCHINGU is possible).
[0535] The prefetch address register 943 is loaded with the physical address of the WORD of the beginning of what kind of pre-planned system. The continuing IBus transaction begins, and when a prefetch counter shows that the **** prefetch of at least one WORD was carried out, the physical address of the beginning of a transaction is compared with the value of the prefetch address. If both match, prefetch data will be used for filling IBus taking over, and a PCI transaction request will start them in the address after the WORD prefetched at the end.
[0536] The prefetch counter 941 is a 4-bit counter, and an increment is carried out to the count same whenever WORD is returned by PMC during prefetch actuation as the depth of the maximum input FIFO. If the continuing IBus transaction matches the prefetch address, a prefetch count is added to an address counter, and it lengthens from a burst counter, and can start in the location where a PCI demand is demanded. If some data with which the IBus transaction was prefetched are needed instead, it is added to the prefetch address which the die length of the demanded burst was lengthened from the prefetch count, and was latched, and the remaining prefetch data will be suspended in order to fill the further demand.
[0537] Data FIFO 944 are asynchronous fall through [8 word x33 bit / FIFO]. The data from PMC are written in FIFO with the bit showing whether data are a part of pre-planned system. As soon as the data from the head of FIFO become usable, reading appearance of them is carried out from FIFO, and they are sent to IBus. The logic which generates a data read-out signal operates synchronizing with clk, and generates an IBus acknowledgement output. When filled using the data with which the transaction was prefetched, the signal from control logic reads a prefetched number of data which is read from FIFO of ******, and gives them to logic.
[0538] Drawing 156 shows the structure of the RBus receiver 914 of drawing 150. The split of the control is carried out between two condition machines 950 and 951. The write-in condition machine 951 controls the interface to RBus. The input address 752 is latched at the starting point of a RBus burst. Each data word of a burst is written in FIFO754 with cutting tool enabling. It will write in, if FIFO954 comes to be full, and r-ready is canceled by logic 951 and an organizer takes care not to write in the WORD beyond it by it.
[0539] The write-in logic 951 notifies initiation of a RBus burst to the Maine condition machine 950 through a resynchronization start signal, and an organizer takes care not to write in the WORD beyond it. The upper address bit which forms a virtual page number is loaded to a counter 957. A virtual page number is sent to MMU and the physical-page number 958 is returned from MMU. A physical-page number and the lower bit of the virtual address are recombined according to a mask, are loaded to a counter 960, and offer the address for the PCI demand to PMC. Clock out of the data for each WORD of a PCI demand and cutting tool enabling is carried out from FIFO954 by the Maine control logic 950 also handling all PMCM interface-control signals. The Maine condition machine shows that it is AKUTIVU through a busy signal, resynchronization of it is carried out to a write-in condition machine, and it is returned.
[0540] The write-in condition machine 951 detects termination of a RBus burst using r-final. Then, loading of the data to FIFO954 is stopped and it notifies that the RBus burst was completed in the Maine condition vessel. The Maine condition machine continues a PCI demand until Data FIFO become empty. And a busy is canceled and a write-in condition machine starts the following RBus burst.
[0541] Return and the memory management unit 915 take charge of the conversion for a physical-page number from a virtual page number again to drawing 150 for the IBus driver (IBD) 912 and the RBus receiver (IBR) 914. The detail of a memory management unit is shown in drawing 157. The conversion look-aside buffer (TLB) 970 of 16 entries receives input data from the TLB address logic 971, and returns an output. The TLB control logic 972 in which the condition machine is contained receives the demand by which the buffer is carried out to TLB address logic from RBR or IBD. If a demand is received, the activity done by the source and TLB of an input will be chosen. Effective TLB activities are a comparison, nullification, all nullification, writing, and read-out. As the source of the TLB input address, there is a register in IBD, a RBR interface (comparison operating), the page table entry buffer 974 (for TLB mistake service), or TLB address logic etc. TLB returns the status of each activity to TLB control logic. The physical-page number from the comparison activity in which it succeeded is returned to IBD and RBR. TLB holds record of the location (LRU) accessed most recently, and for TLB address logic, this is useful, although used as a location of write-in operating.
[0542] When a comparison activity goes wrong, the TLB control logic 972 takes out a signal so that a PCI demand may be started in the page table access control logic 976. The page table address generator 977 generates the PCI address based on a virtual page number using an internal page table pointer register. The data

returned from the PCI demand are latched to the page table entry buffer 974. If the page table entry which matches the virtual address demanded is found, a physical-page number will be sent to the TLB address logic 977, and it will notify that page table access completed the page table access control logic 976 after that. And the TLB control logic 972 writes a new entry in TLB, and starts a comparison activity again.

[0543] The register signal to SCI and the register signal from SCI are 980 by which resynchronization is carried out to both directions. A signal keeps company with all submodules. The module memory interface 981 decodes access to TLB and a page table pointer memory element from a Standard C Bus interface. TLB access is read-only, and in order to obtain data, TLB control logic is used. Both read-out and writing are possible for a page table pointer, and it is direct accessed with a module memory interface. The synchronous circuit is also included in these pass.

[0544] 3.18.11 An example of the peripheral-interface-adapter control section (PIC) of drawing 2 is shown in the peripheral-interface-adapter control-section drawing 158 at the detail. PIC237 -- an external peripheral device -- or it operates by some one of the Mohd which transmits data from a device. fundamental Mohd -- 1 video-outlet Mohd: -- it is this Mohd and data are transmitted to the circumference under control of an external video clock and clock data enabling. PIC237 sends an output clock and a clock enabling sign to the timing needed to output data.

[0545] 2) Video input mode : by this Mohd, data are transmitted to the circumference under control of an external video clock and clock data enabling.

3) Centronics mode : this Mohd is IEEE. According to the standard protocol defined as 1284 criteria, data are transmitted to the circumference from the circumference.

PIC237 separates the protocol of an external interface from an internal data source or the destination if needed. An internal data source writes data in the single stream of output data, and is transmitted to an external peripheral device by Mohd chosen. Similarly, it is used for all the data from the circumference of external filling the transaction which was written in the single input data stream and required of one of the possible in-house-data destinations.

[0546] As the source of possible output data, three, LMC236 (ABus is used), RO249 (RBus is used), and general CBus, are mentioned. PIC237 answers a transaction at once from these data sources only one. The transaction from the one source is thoroughly ended, before the following source is taken into consideration. Generally, only one data source must not become AKUTIVU at any time. When the two or more sources become AKUTIVU, it is processed in order by the priority of CBus, ABus, and RBus.

[0547] Usually, it passes and a module operates under control of the Standard C Bus interface 990 with which the internal register of PIC is contained. Furthermore, the CBus interface 992 can access and control a peripheral device through a co-processor 224. The ABus interface 991 can also process a memory interaction with a local memory control section. In addition to the result organizer 249, both the ABus interface 991 and the CBus interface 992 send data to the output data path 993 in which cutting tool-wide FIFO is contained. Access to an output data path is controlled [which source has a priority or ownership to the output stream, and] by the arbitrator who always checks. An output data path interfaces a bidet with the output-control section 994 and the Centronics control section 997 by which is enabling. Each module 994 and 997 reads 1 byte from the interior FIFO of an output data path at once. The Centronics control section 997 embodies a standard Centronics data interface, in order to control a peripheral device. The logic which controls an output pad is contained in the video outlet control section according to the video outlet protocol demanded. Similarly, the logic which controls any video input criteria used is contained in the video input-control section 998. The video input-control section 998 takes out an output to the input data path unit 999, and this constitutes the data and the cutting tool wide input FIFO which are again written at once in FIFO by the video input-control section 998 or every 1 byte of the Centronics control section 997 by asynchronous.

[0548] Various counters are contained in the data timer 996, and it is used in order to carry out the monitor of the current condition of FIFO in the output data path 993 and the input data path 999. If a co-processor is used, in order to generate simultaneously the multiplex part of a multiplex image or a single image from the above thing, it is considered to be possible to execute the instruction of a duplex stream. A primary instruction stream is used for acquiring the output image of a page now, and in order to begin the rendering of the following page while the primary instruction stream is an idle, it can use a secondary instruction stream. Consequently, in actuation of a canonical mode, the image of a current page is compressed using the JPEG coder 241, after a rendering is carried out. When it is necessary to print an image, a co-processor 241 uses the JPEG coder 241 twice, and thaws a JPEG ENKO dead image. an output device -- since -- it is possible to execute an instruction

for the configuration of the following page or a band between the idle times for which the part of the JPEG DEKO dead image beyond it is not needed. Generally this process raises the rate which generates an image by the overlap of a co-processor of operation. If a co-processor 224 is used especially, in order that a print may be performed and rendering speed may increase as a result by the printer attached to the co-processor, a benefit is obtained in respect of speedup of an image processing activity.

[0549] the above -- probably a suitable example is one operation gestalt of this invention, and it will be clear from the above that correction obvious for this contractor can be performed without separating from the range of this invention.

[0550] Appendix A co-processor microprogramming -- in this chapter, the actuation performed within a co-processor for every activation of a new instruction is explained in full detail. All the self configurations performed by the co-processor between instruction executions are realized by the read/write of an internal register, therefore microprogramming is completely possible for a co-processor by using a PCI bus interface by external C bus interface or an external host. However, becoming difficult from a host synchronous problem generally [in the case of the microprogramming using a host] is expected. The premise of this chapter having information with a reader sufficient in respect of the following about a co-processor is carried out.

1. Activation model 2. instruction set and coding 3. register set 4. internal structure A.1 General matter A1.1 About all instructions other than the general matter control instruction about the setup of a co-processor, and a local DMA instruction, inner data flow is fundamentally put under a pixel organizer's control by the co-processor. The pixel organizer has charge about the decision of the stage when the fetch of the fetch of the head of an input data stream, the count of data, and the last data was carried out. The module of others in a co-processor only answers the sent data fundamentally.

A1.2 The module of all modular configuration sequence is not set up for every instruction. The configuration of some modules is not carried out at all at the time of instruction decoding. Modular configuration sequence is always the order of PO, DCC, OOB, OOC, MDP, JC, RO, and PIC.

A1.3 When the setting-out instruction of other registers is encoded including setting out of a certain register value, the register is set up by the microprogramming which follows in the following order.

1. If a register set otherwise does not exist in a module with the register which should be set up, the register is set up before what kind of other register setting out.

2. When another register set is in a module with the register which should be set up, the register is set up just before _cfg register of the module, after setting out of other registers finishes.

A1.4 Since the operand and the data type of a result are specified, the instruction of many coding of an instruction operand without consistency returns a meaningless result, when other data types are specified. A co-processor opts for a format of the target operand in the following procedure to each operand.

1. When one pixel (a compression cutting tool or incompressible cutting tool) specializes in the internal format of an operand, a corresponding operand organizer is set up reflecting this. The configuration of the data cache controller is not carried out, therefore an operation is continued by normal mode.

2. When "other formats" specializes in the internal format of an operand, a co-processor generates a format of an operand from an instruction. About Operand B and Operand C, it is forward. "Other formats" is not originally specified about Operand A, and behavior of a co-processor is not defined. A corresponding operand organizer becomes a bypass mode, and a data cache controller is set up so that the operand data of the obtained format may be managed. Microprogramming is mutual independence in various inter modules rationally.

A1.5 The execution sequence of syntax and an instruction of a pseudo-instruction is determined by the left end number.

- A register name is Helvetica. It has started with the Bold object.
- The register field is shown by register.field.
- I and D show the instruction word and data word by which the current decryption is carried out, respectively.
- A, B, and C show operand WORD A by which the current decryption is carried out, operand WORD B, and operand WORD C.
- A_deskriptor, B_deskriptor, and C_deskriptor show the descriptor of the data word of the instruction by which the current decryption is carried out.
- R shows WORD as a result of the instruction by which the current decryption is carried out.
- "X:Y" shows connection of X and Y.
- "@X" shows register number X of a co-processor.
- "Cbus (X)" shows activation of the C bus operation X.

- "*Cbus (X)" shows the reception data based on the C bus operation X.
- "*X" shows the virtual memory address X.
- "??" shows an unknown value or an undecided value.
- "set" shows setting out of a data manipulation register.

A.2 Composite operator notes : 1. main operation codes consider that 0xC and 0xD2. ambiguity are the cutting tool (namely, most significant byte) of the top address.
3. The pre multiplication of an accumulator or the operand may be carried out.
4. The non-pre multiplication of the result may be carried out.
5. The instruction length is defined by the number of input pixels.

A.3 Color space conversion notes : 1. input space is always a three dimension. By the default, it is the channel of three least significant pixels. Ambiguity is eliminated.
2. A format of a color table is either among the thing containing one output channel, and the thing containing four output channels.

A.4 JPEG instruction notes : 1. operation code is 0 x2.
2. The register for YATTO is sufficient as Operand C.
3. Many options exist.
/subsampling is performed - don't carry out.
/it filters - don't carry out.
- 1, 3, or four scans.
4. These instructions are related to the register of the shoes set up before an instruction execution.

A.4.1 Expanding notes : the register below 1. needs to be set up before the instruction execution.
- A ro_idr:output image number-of-dimension register and ro_cut : an output cut register and ro_lmt:load limitation register A.4.2 Compression notes: The register below 1. needs to be set up before the instruction execution.
- interval, ro_cut:output cut register, and ro_lmt:load limitation register A.5 of a po_idr:output image number-of-dimension register and a jc_rml:restart marker data coding notes: -- 1. -- all data coding actuation -- compression and compression discharge -- in any case, it is treated similarly. These actuation setting out is almost the same as the time of JPEG.
2. Possible Encoding Actuation, Huffman Coding, and Predicting-Coding 3. -- Possible Decoding Actuation, High-speed Huffman Decryption, Low-speed Huffman Decryption, and Packbits Decryption (Version A)
- packbits decryption (version B)
- The register for setting up is sufficient as the prediction decryption 4. operand C.
5. The following registers need to be set up before the instruction execution.
- A ro_cut:output cut register and ro_lmt : load limitation register A.6 It collapses with conversion and 1. operation code is 0x4 (convolution) and 0x5 (conversion).
2. A co-processor performs actuation which is the superset which is needed for [each] image transformation and image convolution. The only difference between image transformation and the image reefing is that the step size serves as 1 source pixel by the reefing to a kernel step size being the magnitude (a horizontal, perpendicular) of a kernel in image transformation, as far as a co-processor is concerned.
3. Option : the clamp of - final result, wrapping, absolute value 4. notes:conversion, and the reefing cannot be performed [whether are recording of snapping to - contiguity pixel, and a interpolation and a pixel (kernel) is performed, whether the pre multiplication of - source pixel is performed and] in the original location. That is, the content is destroyed when the pointer of a destination is the same as the pointer of the source.

A.7 matrix multiplication notes: -- 1. operation code -- 0x32. option: -- A.8 which may also write [whether the pre multiplication of - source pixel is performed, and] absolute-value[the clamp of - final result, wrapping, and]-izing, and Operand C in a register half toning notes: -- 1. operation code -- a 0x72. option -- the level value of a halftone -- 3. halftone screen -- suitable -- a mesh -- or as long as AMMESSHU, it can carry out to a pixel or a cutting tool.

A.9 memory copy notes: -- 1. operation code -- 0x92. -- this instruction completely uses the device according to individual, in order to complete actuation of a memory copy.
- General-purpose data transfer instruction uses the usual data flow in a co-processor, and can use various functions using the data manipulation unit in PO and RO.
- A peripheral DMA instruction uses the direct connection between PIC and LMC. This means that data manipulation is impossible and an instruction of consecutiveness and a concurrency are possible for it.

A.9.1 General-purpose data transfer A.9.2 Peripheral DMA-transfer notes: 1. concurrency also hopes that that is not right. This is treated by IC.
2. Since 3.PIC which sets up Operand C and as which a register is sufficient is a module treating data, this instruction differs from other "activity" instructions.
A.10 photo CD growth length -- this instruction group consists of three different actuation, i.e., level interpolation, vertical interpolation, and remainder fusion. The setting-out approach of vertical interpolation and remainder fusion is the same. The operation code of all an instruction of these is 0x9.
A.10.1 level interpolation notes: -- 1. pixel or a cutting tool -- receiving -- 2. which can be performed -- the operand of this instruction may be one instruction, and the register to set up is sufficient as Operand C.
A.10.2 Vertical interpolation and remainder fusion notes : setting out of 1. vertical interpolation and remainder fusion is the same.
2. Activation is possible to both a pixel and a cutting tool.
3. The operands of this instruction may be two instructions, and a register set is sufficient as Operand C.
A.11 Control instruction notes : 1. control instruction consists of two kinds of actuation, i.e., a flow control instruction, and an internal access instruction.
A.11.1 Flow control notes : 1. operation code consists of the standby instruction of current, various jump instruction, and various kinds. [instruction / 0xB2. flow control]
3. Within a co-processor, clear installation is not performed and this instruction is not an "activity" instruction. That is, that the submodule in a co-processor performs something actually **** does not carry out like other instructions.
4. The register to set up is sufficient as Operand C.
A.11.2 Internal access (lead)
Notes: In 1. operation code, a 0xA2. lead instruction transmits data besides a co-processor.
3. RO is the only module which performs all within a co-processor actually.
A.11.3 Internal access (light)
Notes: In 1. operation code, a 0xA2. light instruction transmits data in a co-processor.
3. Since this instruction is not an "activity" instruction, perform no modules other than IC actually.
A.12 The reserved instruction notes:1. operation code 0x0 and 0xF are reserved.
2. The reserved instruction issues the error in which a mask is possible.
3. These reserved instructions are to be used as other instructions, when a co-processor is revised from now on.
Appendix-B: register 1.1 A register and table this section explain the register of a co-processor. These registers can be changed by three kinds of approaches.
1. There is an instruction group of a specific co-processor in order to write a ** register. By using these instruction groups, a register is in the memory relevant to a local memory interface using initiation of the PIC bus cycle of an initiator, or the transaction of a general interface, or the R/W from memory is performed.
2. As for many registers, the content changes with the side effects of an instruction execution. The main devices in which a co-processor sets up self for an instruction execution are realized by setting up various registers so that a current condition may be reflected. After instruction-execution termination, each register reflects the condition of a co-processor. Many typical processings are thoroughly specified with a certain instruction, and are set up. It is necessary to set up just before an instruction execution in some registers.
the semantics of "reservation" register bit -- oh, the semantics of "reservation" of a **** register or its component is as follows.
- The data is rejected although the writing to the reserved location can be performed.
- Although reading from the reserved location can be performed, all the registers and register fields where the data is unfixed and that are not pinpointed are "reservation."
1.1.1 The register in the classification co-processor of a register is classified based on the behavior described by this section. these description - external: -- the module exterior (from -- access). It is external access using a CBus interface. That is, PCI in an instruction controller or the target mode by the external CBus interface is used. Notes and a register cannot be set from a PCI bus through BAISETTO mode.
- internal: -- the interior of a module (from -- access)
From the outside, a status-register status register is ReadOnly and can be written from the interior.
KONFIGU 1 register KONFIGU 1 register can be written from the exterior, and is ReadOnly from the interior. KONFIGU 1 register does not support CBus actuation of Type C (that is, bits set mode is not supported), but is used as a register holding cutting tool (or it is bigger than it) configuration information like an address value.

KONFIGU 2 register KONFIGU 2 register can also be written from the exterior, and is ReadOnly from the interior. KONFIGU 2 register supports CBus actuation (namely, bits set mode) of Type C, and is used as a register holding configuration information with the need of setting up by bitwise.
The exterior and the interior to R/W is possible for control 1 register control 1 register. Control 1 register does not support CBus actuation of Type C (that is, bits set mode is not supported), but is a cutting tool (or used as a register holding bigger control information than it.) like an address value.
The exterior and the interior to R/W is possible for control 2 register control 2 register. Control 2 register supports CBus actuation (namely, bits set mode) of Type C, and is used as a register holding control information with the need of setting up by bitwise.
The bit in an interruption register interruption register can be set to 1 from the interior, and can be reset from the exterior to 0 by writing in 1. Module interruption / error register is also this type. Modular interruption/error register consists of the three fields.
[7: 1.1.2 which means all the interruption conditions generated with the [31:24] modules which mean all the exception conditions generated with the [23:8] modules which mean all the error situations (status) generated with 0] module The register map table 1.1 is the register of a co-processor. A number is not the address but a register number.
Table 1.1 Co-processor register 1.1.3 Register definition general-purpose-module register instruction controller register I. An ic_cfgic_cfg register separates into three parts. The lowest cutting tool includes global configuration information. In the 3rd cutting tool, the most significant byte includes the configuration information on Stream B including the configuration information on Stream A from the least significant. The reset value of this register is 0x00000000.
m. is_stat -- this register is divided into four sections. The lowest cutting tool holds the internal state of IC. The 2nd cutting tool holds the result by which the current instruction was decrypted, current, and the prefetched instruction stream from the least significant. The 2nd cutting tool holds all status information about A stream from the most significant. The most significant byte holds the information about B stream. The reset value of this register is 0x00000000.
n. ic_err The register of int ** contains the flag of active and a high which shows whether interruption and an error occurred inside IC. Each bit is cleared by writing in 1.
o. ic_err_int_en -- a reset value is 0x00000000 including the mask of authorization of the error with these various registers, or interruption.
p. ic_ipa -- this register holds 30 bits of most significants of the virtual address used for the instruction fetch Of Stream A. The two least significant bits are assumed by 0 noting that the instruction should have aligned. The reset value of this register is 0x00000000.
q. ic_tda -- this register holds the "to do" value Of Stream A. This is a 32 bits (wrapping) sequence number until a proper instruction exists. The reset value of this register is 0x00000000.
r. ic_fna -- this register holds "termination" value Of Stream A. This shows the instruction completed at the end by the 32 bits (wrapping) sequence number. The reset value of this register is 0x00000000.
s. ic_inta -- this register holds "interruption" number Of Stream A. This is that 32 bits (wrapping) sequence number which applies interruption to where, when a device is effective and is prepared. The reset value of this register is 0x00000000.
t. ic_loa -- this register holds the 32 bits (wrapping) sequence number of a duplication instruction of the last performed by Stream A. The reset value of this register is 0x00000000.
u. ic_ipb -- this register holds 30 bits of most significants of the virtual address used for the instruction fetch Of Stream B. The two least significant bits are assumed by 0 noting that the instruction should have aligned. The reset value of this register is 0x00000000.
v. ic_tdp -- this register holds the "to do" value of Stream B. This is a 32-bit (wrapping) number until a proper instruction exists. The reset value of this register is 0x00000000.
w. ic_fnb -- this register holds "termination" value Of Stream B. This shows the instruction completed at the end by the 32 bits (wrapping) sequence number. The reset value of this register is 0x00000000.
x. ic_intb -- this register holds "interruption" number of Stream B. This is that 32 bits (wrapping) sequence number which applies interruption to where, when a device is effective and is prepared. The reset value of this register is 0x00000000.
y. ic_lob -- this register holds the 32 bits (wrapping) sequence number of a duplication instruction of the last performed by Stream B. The reset value of this register is 0x00000000.

z. ic_sema -- this register is the alias which used the side effect of an ic_stat register, this register reads and ** is the side effect of a demand of the register semaphore Of Stream A.
aa. ic_semb -- this register is the alias which used the side effect of an ic_stat register, and reading of this register is the side effect of a demand of the register semaphore of Stream B.
Input interface register ab. iis_cfgac. iis_statad. iis_err_intae. iis_err_int_enaf. iis_ic_addrag. iis_dcc_addrah. iis_po_addrai. iis_burstaj. iis_base_addrak. iis_test external-interface controller register al. eic_cfgam. eic_statan. An error and interruption bit of eic_err_inteic_err_int register can be set up only by EIC, and can be reset only with software. The usual error and an interruption bit are reset by writing 1 in the bit. The error bit which is the copy of a PCI configuration register bit must be cleared by writing in a PCI configuration register. That is, no copies in eic_err_int influence.
ao. eic_err_int_enap. eic_testaq. eic_pobar. eic_high_addras. eic_wtlb_vat. eic_wtlb_pau. eic_mmu_v notes: By a page fault error or the error of MMU to a PCI bus, MMU can change the value of this register at any time, if not invalid.
av. eic_mmu_p notes: By a page fault error or the error of MMU to a PCI bus, MMU can change the value of this register at any time, if not invalid.
aw. eic_ip_addr notes: By the error to a PCI bus from IBus, IBD can change the value of this register at any time, if not invalid.
ax. eic_rp_addr notes: By the error to a PCI bus from RBus, RBR can change the value of this register at any time, if not invalid.
ay. eic_ig_addr notes: By the error of a general-purpose bus, GBC can change the value of this register at any time, if not invalid.
az. eic_rg_addr notes: By the error of a general-purpose bus, GBC can change the value of this register at any time, if not invalid.
The alias of the PCI bus configuration space which consists of 16 words of aliases of PCI bus configuration space is carried out to the register shown in the address from 0xc0 to 0xcf.
local memory controller register ba. lmi_cfg -- this register contains many the configuration bits and control bits which are used for determining the processing mode and parameter of LMC. When sdram_1 pin is a high, the bit which refers to SDRAM processing specially does not have effect at all. This register has the reset value 0x20000100 which is refresh spacing of 3.2 microseconds, when the frequency of clkin is 80MHz. All special Mohd and functions are invalid to a power up, and all access permissions are equally set as 0. Although refresh is effective at the time of reset, other modules are invalids (E= 0). Refresh is not influenced by E bits.
bb. A lmi_stat status register consists of a modular active ** undecided bit like the information inside a machine. For are driving the state machine with the twice as many clock as a CBus interface, therefore holding the status information of each newest 80MHz clock two cycle, it is 2 field need.
bc. The status register of a lmi_err_int error and interruption interrupts and holds an exception and the information on an error situation. A register can be written, reading returns status information and the writing of 1 to a specific bit resets the bit. The writing of 0 does not have effect at all to the bit.
This register must have the reset value 0x00000000, and this shows that interruption and an error have not occurred. A reservation bit is always 0 and can never change a condition.
bd. A lmi_err_int_en register error, an exception, and an interruption effective register are used for selection of the validity of an error and an exception interruput signal, and an invalid. A register can be written. This register is used for coming into effect by bitwise based on the error in a lmi_err_int register, an exception, and each interruption. The response of 1 to 1 is between the bit of this register, and the bit of a lmi_err_int register. the bit of the specification of a lmi_err_int_en register -- yes -- the bit to which a lmi_err_int register corresponds if it comes to be alike -- effective -- becoming -- it -- yes, come out and it is -- if it becomes, a LMC module error, an exception or an interruput signal, c_err, c_exp, or c_int can be generated. The bit to which the specific bit of a lmi_err_int_en register is cleared and a lmi_err_int register corresponds in drops can become an invalid, and c_err, c_exp, or c_int cannot be generated. Since it is unexceptional to LMC, the exp_mask bit of this register does not influence at all, but is reservation altogether. The reset value of this register is 0x00000000 which makes an invalid all errors and sources of interruption. The bit which is not used is always 0 and cannot be set to a high.
be. lmi_dcfg -- this configuration register holds the design parameter which determines the size and the configuration in the case of using a DRAM chip. This register holds reset value 0x0007ff80 which makes the value of all timing limits maximum.

bf. lmi_mode The configuration register of register ** holds the information written in a SDRAM mode register as part of initialization processing. You can always write this register and may also write it in SDRAM by setting an initialization bit. This register has the reset value 0x0037. This useful default is promptly required after powering-on precharge or reset of level 1. This reads, sets delay as three clocks, and sets burst length as the full page using a sequential lap. An initialization bit is set in order to program a SDRAM mode register in first stage after all reset, if sdram_1 pin is a low. This bit is automatically cleared by zero after write-in activation of a mode register.
Peripheral-interface-adapter register bg. pic_cfg Register bh. pic_statbi. An error and interruption bit of a pic_err_intpic_err_int register are set by only PIC, and are reset only by software. Each bit is bj. reset by writing in 1. pic_err_int_enbk. pic_abus_cfgbl. pic_abus_addrbm. The pic_cent_cfgpic_cent_cfg register includes reading / write-in signal, and the ReadOnly status signal which control the aspect of affairs of all interfaces, when the Centronics mode is effective.
bn. pic_cent_dirbo. pic_reverse_cfgbp. pic_timer0bq. pic_timer1 data-cache controller register br. dcc_cfg1bs. dcc_cfg2bt. dcc_statbu. dcc_err_intbv. dcc_err_int_enbw. dcc_lv0bx. dcc_lv1by. dcc_lv2bz. dcc_lv3ca. dcc_addrcb. dcc_raddrbcc. dcc_raddrccd. dcc_test operand organizer register : to which two same operand organizers exist in an operand organizer register -- they are the operand organizer B and the operand organizer C. These two registers for operand organizers are described here.
ce. oon_cfg(oob_cfg=0x70,ooc_cfg=0x80
cf. oon_stat(oob_cfg=0x71,ooc_cfg=0x81)
cg. oon_err_int(oob_err_int=0x72,err_int=0x82)
ch. oon_err_int_en(oob_err_int_en=0x73,err_int_en=0x83)
ci. oon_dmr(oob_dmr=0x74,ooc_dmr=0x84
cj. oon_subst(oob_subst=0x75,ooc_subst=0x85)
ck. oon_cdp(oob_cdp=0x76,ooc_cdp=0x86
cl. oon_len(oob_len=0x77,ooc_len=0x8cm. oon_said(oob_said=0x78,ooc_said=0x88)
cn. oon_tile(oob_tile=0x79,ooc_tile=0x89)
Pixel organizer register co. po_cfgcp. po_statcq. po_err_intcr. po_err_int_encs. po_dmrct. po_substcu. po_cdpcv. po_lencw. po_saidcx. po_idrcy. po_muv_validcz. po_muv main data path register da. mdp_cfg All bits are reset by 0.
db. The bit of all mdp_stat is reset by 0.
dc. The bit of all mdp_err_int is reset by 0.
dd. The bit of all mdp_err_int_en is reset by 0.
de. mdp_test All bits are reset by 0.
df mdp_op1 All bits are reset by 0.
dg mdp_op2 All bits are reset by 0.
dh mdp_por All bits are reset by 0.
di mdp_bi All bits are reset by 0. A mdp_bi register is used for various Mohd's various things.
dj mdp_bm All bits are reset by 0. A mdp_bm register is used for that from which different Mohd differs.
dk mdp_len All bits are JPEG encoder registers reset by 0. dl jc_cfgdm jc_statdn jc_err_intdo jc_err_int_endp jc_rsidq jc_decodedr jc_resds jc_table_sel result organizer register dt ro_cfgdu ro_statdv ro_err_intdw ro_err_int_endx ro_dmrdy ro_substdz ro_cdpea ro_leneb ro_saec ro_idred ro_vbaseee ro_cutef Alias of ro_lmtPCI configuration space PCI configuration space 256 bytes, It is the block of the register defined by PCI, and a host does the configuration of the PCI device, reads the condition, or admits carrying out. It is accessed using a PCI configuration cycle. The mirror of the register is carried out to the ReadOnly area of the internal memory of a co-processor again, therefore it can be read using the usual memory cycle of PCI. A format of the configuration space mounted in EIC is shown in a table 1.141.1.
Table 1.141.1 The bit of the reservation in the register of spatial layout reservation of a co-processor PCI configuration and the mounted register returns 0 to reading, and does not influence by writing. The address of the configuration space of the range of 0x40-0xff is also reservation. The configuration register only for vendors is not defined.
eg The register of vendor ID ** is ReadOnly. The vendor ID of CISRA is 0x11AC.
eh The register of device ID ** is ReadOnly. The device ID of a co-processor is 0x0001. 8 bits of :most significant by which the device ID field is divided into the two 8-bit fields are the number (0x0 is a co-processor) which shows ****** of a device, and the least significant 8 bits show the version number (0x1 is the

version of a co-processor) of the device.
ei The definition of the field of a command register command register is shown in a table 1.142. All the bits of this register that are not reserved can do reading/writing. This register is set to 0x0000 after reset.
ej Status register The definition of the field of a status register is shown in a table 1.143. Reading of this register is usually a passage. Some bits of this register are ReadOnly. Other bits are set to 1 by only the co-processor, and are reset by only the host 0 (except for a static test mode). The writing of;0 which a host is writing 1 in the bit, and is reset does not make semantics. This register is set to 0x0280 after reset.
ek Revision ID This is the register of ReadOnly. The initial revision ID of a co-processor is 0x01. el Class code This is the register of ReadOnly. Since a co-processor is not suitable for the class code by which PCISIG was defined, this register is set to 0xFF0000.
em cache linesize -- this is a register which determines the cache linesize of a system by the 32-bit word unit and which can be written. This is determined when a co-processor uses a memory reading line and a memory multiplex reading command. A co-processor supports the values from 0 to 255 of this register. 0 in this register makes an invalid the format of a memory reading line and memory multiplex reading. This register is set to 0x00 at the time of reset.
en The register of delay timer ** is a register which specifies the maximum number of clocks which CPU uses for processing of all PCI and which can be written. A co-processor supports the value of 0 to 255 in this register. This register is set to 0x00 at the time of reset.
eo The register of ReadOnly of header type ** is set to 0x00. This means that a co-processor uses the configuration space of the layout of Type 0.
ep The register which can write base address ** is used in order to arrange the internal register of a co-processor, an internal memory, a local memory, and a general interface in a host's memory map. Various resources of a co-processor can occupy 64MB (not all are used), therefore can write in only 6 bits of heads of this register. Connection of all the remaining bits is carried out to 0 in hard. 4 bits of the low order of this register are a control bit of ReadOnly, and connection also of these is carried out to 0. This means that a register refers to room and a co-processor is mapped anywhere in the 32-bit space by the side of a host, and when the resource of a co-processor is a target, it means that it cannot prefetch.
eq the ReadOnly register of subsystem vendor ID ** -- a host -- a system -- mounting -- it enables it to identify the vendor of a <DP N=0225> **** PCI board (as opposed to the vendor of the component mounted in the PCI interface on a board) The content of this register uses an EIC configuration serial port at the time of reset, and is loaded to it.
er The ReadOnly register of subsystem ID ** enables it to identify the PCI board on which the host was mounted in the system. The content of this register uses an EIC configuration serial port at the time of reset, and is loaded to it. This mechanism makes possible coding and reading from a host from the outside of information required for the function or configuration of a board.
es interruption Rhine -- this register that can be written is used in order that the system software may enable it to record interrupt line routing information, and it can be accessed with interruption service software. Processing in a co-processor is not affected at all. This register is set to 0x00 at the time of reset.
et Connection of the ReadOnly register of interruption pin ** is carried out to 0x01 in hard. This shows that a co-processor uses the inta_1 interruption pin of PCI.
eu Min_Gnt -- this ReadOnly register shows a host the burst interval length of a 1/4-microsecond unit which a co-processor requires. The optimal value of this register was not decided yet.

Since it became timeout time, translation result display processing is stopped.

Drawings are not displayable due to the volume of the data (more than 200 drawings).

* NOTICES *

1.This document has been translated by computer. So the translation may not reflect the original precisely.
2.**** shows the word which can not be translated.
3.In the drawings, any words are not translated.

DESCRIPTION OF DRAWINGS

[Brief Description of the Drawings]
[Drawing 1] Drawing showing actuation of the raster image co-processor within a host computer environment
[Drawing 2] Drawing having shown the raster image co-processor of drawing 1 in the detail more
[Drawing 3] Drawing showing the memory map of a raster image co-processor
[Drawing 4] Drawing showing the relation between co-processors the result in CPU, an instruction queue, an instruction operand, and a shared memory
[Drawing 5] Drawing showing the relation between the instruction generation section, the memory management section, the queue Management Department, and a co-processor
[Drawing 6] Drawing showing actuation of the graphics coprocessor which reads an instruction from a pending instruction queue and is arranged at a termination instruction queue
[Drawing 7] Drawing explaining the need of standing by until fixed-length patrol buffer mounting of an instruction queue is shown and a buffer overflows
[Drawing 8] Drawing showing the instruction-execution stream used in a co-processor
[Drawing 9] Instruction-execution flow chart,
[Drawing 10] Drawing showing the standard instruction word format used in a co-processor
[Drawing 11] Drawing showing the instruction word field of a standard instruction
[Drawing 12] Drawing showing the data word field of a standard instruction
[Drawing 13] Drawing showing the instruction control section of drawing 2 typically
[Drawing 14] Drawing having shown the execution control section of drawing 13 in the detail more
[Drawing 15] The state transition diagram of an instruction control section
[Drawing 16] Drawing showing the instruction decode section of drawing 13
[Drawing 17] Drawing having shown the instruction sequencer of drawing 16 in the detail more
[Drawing 18] The state transition diagram of ID sequencer of drawing 16
[Drawing 19] Drawing having shown the prefetch buffer control section of drawing 13 in the detail more
[Drawing 20] Drawing showing the standard format of the register memory used by the co-processor, and inter module relation
[Drawing 21] Drawing showing a format of the control bus processing used in a co-processor
[Drawing 22] A co-processor is drawing showing inner data flow a part.
[Drawing 23] Drawing showing various examples of data reformatting used in a co-processor
[Drawing 24] Drawing showing various examples of data reformatting used in a co-processor
[Drawing 25] Drawing showing various examples of data reformatting used in a co-processor
[Drawing 26] Drawing showing various examples of data reformatting used in a co-processor
[Drawing 27] Drawing showing various examples of data reformatting used in a co-processor
[Drawing 28] Drawing showing various examples of data reformatting used in a co-processor
[Drawing 29] Drawing showing various examples of data reformatting used in a co-processor
[Drawing 30] Drawing showing the format conversion performed in a co-processor
[Drawing 31] Drawing showing the format conversion performed in a co-processor
[Drawing 32] Drawing showing input data transform processing performed in a co-processor
[Drawing 33] Drawing showing various data conversion performed in a co-processor
[Drawing 34] Drawing showing various data conversion performed in a co-processor

[Drawing 35] Drawing showing various data conversion performed in a co-processor
[Drawing 36] Drawing showing various data conversion performed in a co-processor
[Drawing 37] Drawing showing various data conversion performed in a co-processor
[Drawing 38] Drawing showing various data conversion performed in a co-processor
[Drawing 39] Drawing showing various data conversion performed in a co-processor
[Drawing 40] Drawing showing various data conversion performed in a co-processor
[Drawing 41] Drawing showing various data conversion performed in a co-processor
[Drawing 42] Drawing showing output-data conversion from various interior performed in a co-processor
[Drawing 43] Drawing showing various examples of data conversion performed in a co-processor
[Drawing 44] Drawing showing various examples of data conversion performed in a co-processor
[Drawing 45] Drawing showing various examples of data conversion performed in a co-processor
[Drawing 46] Drawing showing various examples of data conversion performed in a co-processor
[Drawing 47] Drawing showing various examples of data conversion performed in a co-processor
[Drawing 48] Drawing showing various fields used with the internal register which determines which data conversion should be used
[Drawing 49] The block diagram of a graphics subsystem using the data normalization
[Drawing 50] The circuit diagram of data normalization equipment
[Drawing 51] Drawing showing the pixel processing performed in synthetic processing
[Drawing 52] Drawing showing the instruction word format for synthetic processing
[Drawing 53] Drawing showing the data word format for synthetic processing
[Drawing 54] Drawing showing the instruction word format for tile processing
[Drawing 55] Drawing showing actuation of the tile instruction to an image
[Drawing 56] Drawing showing utilization processing of the color section / location table within the section for carrying out remapping of the color value
[Drawing 57] Drawing showing the storing format of the section / location table within the section in the MUV buffer of a co-processor
[Drawing 58] Drawing showing color transform processing using the interpolation performed in a co-processor
[Drawing 59] Drawing showing improvement processing of color transform processing in the edge performed in a co-processor
[Drawing 60] Drawing showing the color space conversion processing for 1 output color performed in a co-processor
[Drawing 61] Drawing showing memory storing within the cache of the co-processor when using a single color output color space conversion
[Drawing 62] Drawing showing the technique used by two or more color space conversions
[Drawing 63] Drawing showing the address remapping processing for the cache used in two or more color space conversion processing
[Drawing 64] Drawing showing the instruction word format in a color space conversion instruction
[Drawing 65] Drawing showing the two or more color conversion technique
[Drawing 66] Drawing explaining generation of MCU in JPEG transform processing performed by the co-processor
[Drawing 67] Drawing explaining generation of MCU in JPEG transform processing performed by the co-processor
[Drawing 68] Drawing showing the structure of the JPEG coding section of a co-processor
[Drawing 69] Drawing showing the quantization section of drawing 68 in a detail more
[Drawing 70] Drawing showing the Huffman coding section of drawing 68 in a detail more
[Drawing 71] Drawing showing the Huffman coding section and the decode section
[Drawing 72] Drawing showing the Huffman coding section and the decode section
[Drawing 73] Drawing explaining deletion / constraint processing of the JPEG data used by the co-processor
[Drawing 74] Drawing explaining deletion / constraint processing of the JPEG data used by the co-processor
[Drawing 75] Drawing explaining deletion / constraint processing of the JPEG data used by the co-processor
[Drawing 76] Drawing showing an instruction word format of a JPEG instruction
[Drawing 77] The block diagram of common discrete cosine transform equipment (conventional example)
[Drawing 78] Drawing showing the arithmetic data path of the DCT equipment of the conventional example
[Drawing 79] The block diagram of the DCT equipment used by the co-processor

[Drawing 80] The block diagram showing the arithmetic circuit of drawing 79 in a detail more
[Drawing 81] Drawing showing the arithmetic data path of the DCT equipment of drawing 79
[Drawing 82] Drawing showing the typical Huffman coding data with which the interleave of the bit field (that by which cutting tool alignment shall not be carried out) which is not encoded like a JPEG format was carried out
[Drawing 83] Drawing having shown more the structure of the whole Huffman decode section of the JPEG data of drawing 84 in the detail
[Drawing 84] Drawing showing the structure of the whole Huffman decode section of JPEG data
[Drawing 85] Drawing in which showing data processing under stripper block which deletes the bit field by which cutting tool alignment was carried out, and which is not encoded from input data, and also showing the example of the tag sign corresponding to the data outputted from a stripper
[Drawing 86] Drawing showing the configuration and data flow of a data pre shifter
[Drawing 87] Drawing showing the control logic of the decode section of drawing 81
[Drawing 88] Drawing showing the configuration and data flow of a marker pre shifter
[Drawing 89] The block diagram of the combinational circuit which decodes a Huffman-coding value in JPEG coding,
[Drawing 90] The concept of a padding field, and the block diagram of the decode section of a padding bit
[Drawing 91] Drawing showing the example of the data format which is outputted from the decode section and used in a co-processor
[Drawing 92] Drawing showing the technique used in an image transformation instruction
[Drawing 93] Drawing showing the instruction word format in an image transformation instruction
[Drawing 94] Drawing showing a format of the image transformation kernel used by the co-processor
[Drawing 95] Drawing showing a format of the image transformation kernel used by the co-processor
[Drawing 96] Drawing showing utilization processing of the index table for the image transformation used by the co-processor
[Drawing 97] Drawing showing the data field format for the instruction used by conversion or the reefing,
[Drawing 98] The explanatory view of bp field of instruction word
[Drawing 99] Drawing showing the reefing processing used by the co-processor
[Drawing 100] Instruction word format drawing of the reefing instruction used by the co-processor
[Drawing 101] Instruction word format drawing of matrix multiplication used by the co-processor,
[Drawing 102] Drawing showing hierarchical image actuation processing in which it is used by the co-processor
[Drawing 103] Drawing showing hierarchical image actuation processing in which it is used by the co-processor
[Drawing 104] Drawing showing hierarchical image actuation processing in which it is used by the co-processor
[Drawing 105] Drawing showing hierarchical image actuation processing in which it is used by the co-processor
[Drawing 106] Drawing showing the instruction word sign in a hierarchical image instruction
[Drawing 107] Drawing showing the instruction word sign of the flow control instruction used by the co-processor
[Drawing 108] Drawing showing a pixel organizer in a detail more
[Drawing 109] Drawing showing the operand fetch section in a pixel organizer in a detail more
[Drawing 110] Drawing showing the various storing formats used by the co-processor
[Drawing 111] Drawing showing the various storing formats used by the co-processor
[Drawing 112] Drawing showing the various storing formats used by the co-processor
[Drawing 113] Drawing showing the various storing formats used by the co-processor
[Drawing 114] Drawing showing the various storing formats used by the co-processor
[Drawing 115] Drawing showing more the MUV address-generation section in the pixel organizer of a co-processor in a detail
[Drawing 116] The block diagram of the multiplex value (MUV) buffer used by the co-processor
[Drawing 117] Drawing showing the structure of the encoder of drawing 116
[Drawing 118] Drawing showing the structure of the decoder of drawing 116
[Drawing 119] Drawing showing the structure of the address-generation section of drawing 116 which reads in

JPEG mode (pixel decomposition) and generates the address
[Drawing 120] Drawing showing the structure of the address-generation section of drawing 116 which reads in JPEG mode (pixel restoration) and generates the address
[Drawing 121] Drawing showing the configuration of a memory module equipped with the store of drawing 116
[Drawing 122] Drawing showing the structure of the circuit which multiplexes the read-out address to a memory module
[Drawing 123] Drawing showing how a look-up table entry is stored in the buffer which operates in single look-up table mode
[Drawing 124] Drawing showing how a look-up table entry is stored in the buffer which operates in multiplex look-up table mode
[Drawing 125] Drawing showing how a pixel is stored in the buffer which operates in JPEG mode (pixel decomposition)
[Drawing 126] Drawing showing how a single color is stored from the buffer which operates in JPEG mode (pixel restoration)
[Drawing 127] Drawing showing an organizer's structure in a detail more as a result of a co-processor
[Drawing 128] Drawing showing the structure of the operand organizer of a co-processor in a detail more
[Drawing 129] The block diagram of the computer architecture for the main data path section used in a co-processor
[Drawing 130] It is the block diagram of reception and the input interface for storing and carrying out a rearrangement about an input data object because of the further processing.
[Drawing 131] The block diagram of the image data processor for performing arithmetic operation to an input data object
[Drawing 132] The block diagram of the color channel processor for performing arithmetic operation to one channel of an input data object
[Drawing 133] The block diagram of the multifunctional block in a color channel processor
[Drawing 134] The block diagram for synthetic actuation
[Drawing 135] Drawing showing the inverse transformation of a scan line
[Drawing 136] The block diagram of a step required in order to calculate the value in the specified pixel
[Drawing 137] The block diagram of an image transformation engine
[Drawing 138] Drawing showing two formats in kernel desk lip SHON
[Drawing 139] Drawing showing a definition and interpretation of bp field
[Drawing 140] The block diagram of the multiplication and adder unit which performs matrix multiplication
[Drawing 141] Drawing showing the control in the cache and cache control section in a co-processor, the address, and data flow
[Drawing 142] Drawing showing the memory configuration of a cache
[Drawing 143] Drawing showing the address format for the cache control section in a co-processor
[Drawing 144] The block diagram of the multifunctional block in a color channel processor
[Drawing 145] Drawing showing the co-processor input interface switch of the cache of drawing 144, and a cache controller
[Drawing 146] Drawing showing 4 port dynamic local memory control section of the co-processor which shows the main address and a data path
[Drawing 147] The condition organization chart in the control section of drawing 146
[Drawing 148] Drawing showing the pseudo code which listed the function in the arbitration section of drawing 146 in the detail
[Drawing 149] Drawing showing the structure and the vocabulary of a claimant priority bit which are used in drawing 146
[Drawing 150] Drawing showing more the external-interface control section in a co-processor in a detail
[Drawing 151] Drawing showing the mapping processing to the physical address used by the co-processor, or the mapping processing from a physical address
[Drawing 152] Drawing showing the mapping processing to the physical address used by the co-processor, or the mapping processing from a physical address
[Drawing 153] Drawing showing the mapping processing to the physical address used by the co-processor, or the mapping processing from a physical address

[Drawing 154] Drawing showing the mapping processing to the physical address used by the co-processor, or the mapping processing from a physical address
[Drawing 155] Drawing showing the IBus receive section in drawing 150 in a detail more
[Drawing 156] Drawing showing the RBus receive section in drawing 2 in a detail more
[Drawing 157] Drawing showing the memory management section in drawing 150 in a detail more
[Drawing 158] Drawing showing more the peripheral-interface-adapter control section in drawing 2 in a detail

[Translation done.]